CSC/TM-87/6728

EXPERT SYSTEM METHODOLOGY STUDY REPORT

Prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Goddard Space Flight Center
Greenbelt, Maryland

By

COMPUTER SCIENCES CORPORATION
4600 Powder Mill Road
Beltsville, Maryland

Under

CONTRACT NAS 5-27888
Task Assignment 357

December 1987

Prepared by:

John H. Baumert 12/29/87
J. Baumert Date

Anna R. Critchfield 12/29/87.
A. Critchfield Date

Karen S. Leavitt 12/30/87
K. Leavitt Date

Approved by:

G. Maury for 12/30/87
G. Maury Date
Technical Supervisor

L. Botten 12/30/87
L. Botten Date
Functional Area Manager

ACKNOWLEDGMENTS

The authors would like to take this opportunity to express their gratitude to those who allowed us to interrupt their work schedules to be interviewed for this study: LeRoy Botten, Jon Buser, Joy Bush, Bikas Das, Tom Davis, Lew Gilstrap, Robert Jackson, Craig Knoblock, Kelly Lindenmayer, Ed Medeiros, Glenn Miller, Keith Richardson, Andrea Weiss, and Bill Woodard. They made a valuable contribution to the study.

ABSTRACT

This Expert System Development Methodology Study Report documents the results of a study of approaches used to develop expert systems. The research methods included a review of relevant literature; interviews of individuals active in expert system development; and participation in conferences and seminars related to artificial intelligence, expert systems, and software development. The study was completed on November 15, 1987.

Specific expert system development methodologies are reviewed as well as the traditional "waterfall" method with regard to their applicability to the NASA/Goddard Space Flight Center environment. The report includes requirements for an expert system development methodology. Synopses of the literature reviewed are included in an appendix.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

SECTION 1 - INTRODUCTION

This section describes the purpose and scope of the study and provides some necessary background information to ensure that all readers approach the study and its results from a common reference point. The background information explains the impetus for the expert system development methodology study. This is followed by definitions of common terminology as applied to expert system development in this report. This section concludes with a description of the organization of the report.

1.1 PURPOSE OF THE STUDY

The purpose of the expert system development methodology study is to identify and evaluate existing or proposed methods used in the development of expert systems. In particular, the study assesses approaches to knowledge acquisition, knowledge representation, implementation, verification, delivery, maintenance, enhancement, quality assurance, and configuration control of an expert system from a technical perspective. The study also reviews existing approaches to expert system management. In assessing existing methodologies and in defining the necessary elements of an expert system methodology, task members worked from the following premises. First, a methodology is essential to developing an operable and maintainable system. Second, a good methodology is an aid to developers. To be good, a methodology, whatever its details, must allow developers to produce quality software products and must not constrain them in their creativity or innovation. Finally, a good methodology is an aid to management, providing guidance in planning a project and establishing mechanisms for monitoring and controlling it.

1.2 BACKGROUND

CSC and NASA have many years of experience working together on software development. Their successful, long-term relationship has led to a mutually understood and agreed upon approach to software development. This approach includes the management aspects of planning, monitoring, and controlling the development process, as well as the technical aspects involved in the actual development of software systems.

Soon after starting an assignment to develop the prototype expert system Platform Management System (PMS) Resource Envelope Scheduling System (PRESS), CSC realized that the traditional management and technical approaches to software development could not meet the needs of expert system development. CSC also recognized that the documented expert system development approaches as represented in the literature and agreed to by both developers and customers, while seemingly appropriate, could not be readily applied in a real-world setting.

From the experience gained in developing PRESS and through conversations with other expert system developers, both government and contractors alike, CSC identified a number of problems that pervade expert system development, including

- Difficulty in scheduling and planning to allow for radical changes to the system
- Difficulty in communicating between/among experts and knowledge engineers
- Difficulty in modularizing a production rule base
- Difficulty in quantifying and measuring progress and status
- Difficulty in devising a mechanism for validating and verifying the product

- Lack of guidance in the role of and mechanisms for quality assurance and configuration management of expert systems

The current task, therefore, grew out of the realization that a comprehensive methodology that covers all aspects of expert system development is needed.

1.3 TERMINOLOGY

A pervasive problem in the field of expert system development, and one encountered in this study, is lack of agreement on terminology. An expert system means something different to nearly everyone. For the purpose of this study, an expert system is defined as "a computer system designed to simulate the problem-solving behavior of a human who is expert in a narrow domain" [11]. This definition from Denning is precise enough to convey the philosophy of expert system technology yet broad enough to allow different approaches to expert system development. This definition provided a common starting point for discussions among task members, with interviewees, and in assessing literature. It also provides a similar degree of commonality for readers of this report.

Similarly task members used a common definition for prototyping. Prototyping, an important element of expert system development, has a number of connotations which depend on the background of the interpreter. In this study, a prototype is an early working model of a proposed system that does not exhibit all the system's features. For an expert system, a series of prototypes is used to establish requirements (i.e., system behavior) as well as to demonstrate proof of concept. Unlike a hardware prototype, an expert system prototype may or may not be discarded. If the prototype is a successful proof of concept, the design

concept is retained for the next iteration. If the prototype fails to satisfy the users and developers, it may be discarded or substantially revised. In essence, the requirements and the design of the expert system evolve through the prototyping process. The final prototype defines the requirements of the functional expert system; these requirements are then implemented to meet performance and other operational requirements. The expert system, therefore, is derived from the knowledge and experience gained from the prototypes and not from forcing each prototype to fit within artificial constraints dictated by its predecessor. The prototyping process also leads to a more robust expert system because poor designs have been eliminated and missing requirements added.

A broad definition of tool is used in this study. Tools consist of hardware and software packages used to assist expert system development. These include languages, shells, knowledge acquisition programs, and hardware devices that host artificial-intelligence software packages. Requirements is used to represent the elements that must be present in the system and that act in harmony with each other to satisfy the customer's needs. For an expert system, this may include a subset of the expert's knowledge. The expert's knowledge is the pool of information and the physical and mental skills possessed by the expert that allows that person to be recognized as an expert in that area.

Finally, a general definition of a methodology is considered necessary. A methodology is the framework that encompasses specific techniques and guidelines for defining and accomplishing work from the beginning through the end of the project. It also provides guidance in planning and organizing to meet the requirements of the task, and it

identifies the controls that must be implemented to monitor and assess the quality of the product and the progress toward completion. Perhaps most importantly, a methodology provides a common frame of reference for all engineers, developers, and managers, and for the customer.

1.4 ORGANIZATION OF THE REPORT

The remainder of the report documents the results of this study. It is divided into four main areas. Section 2 describes the types of research comprising the study. Section 3 analyzes the information learned during this research. Section 4 presents recommendations as to the essential elements that must be included in any methodology proposed for expert system development. Finally, Section 5 contains a brief summary of the conclusions and suggestions for the future.

SECTION 2 - EXPERT SYSTEM DEVELOPMENT METHODOLOGY RESEARCH APPROACH

The goal of this study is to identify and evaluate existing or proposed methods used to develop expert systems. The initial assumption was that the study would consist primarily of a literature search to provide suitable descriptions of various methods and the subsequent evaluation and critique of the methods. This initial assumption was proven invalid. Because expert systems represent a relatively new field in computer science, few systems have completed their life cycle. Accordingly, documented experiences with the full range of an expert system development are necessarily limited.

The literature search proved that essentially no documented, comprehensive, formal methodology, as defined in Section 1, exists for developing expert systems. This is especially true for a formalism describing the development of an expert system from its inception through operations and maintenance. (However, CSC expects this situation to change within the next year. Interest in a formal methodology is high, as evidenced by discussions at conferences and by several very recent papers that discuss the need for a formal methodology.)

This finding indicated that the direction of the research needed to shift. To determine whether an undocumented methodology existed, interviews were added as a research method. The interviews essentially confirmed the results of the literature search. As one interviewee phrased it, "management is a little loose." Furthermore, the interviewees discussed expert systems in general terms, admitting that their personnel followed no formal procedures and that management controls were noticeably lacking.

Conferences were also added to the task agenda. The conferences proved beneficial in that they exposed task members to the most recent advances in the field of expert system development. Hence, what began as a literature search evolved into a broader three-tiered research project, which painted a comprehensive picture of the state of expert system development and confirmed the need for a full-scale methodology.

This three-tiered approach balanced a carefully selected literature review with interviews of expert system practitioners and with attendance at conferences and seminars to determine the latest in expert system development methodology. This section describes the three approaches--literature search, interviews, and conferences attended.

2.1 LITERATURE SEARCH

The initial research effort focused on the available literature. This approach helped familiarize task members with current ideas and development methods, and provided a pool of names for subsequent interviews. Appendix A presents a bibliography of the literature expanded to include a synopsis of each item read. Appendix A is arranged alphabetically by author to facilitate general use. References are also provided at the end of this report and are arranged numerically by a task-assigned literature identifier that is used to reference the documents in this report.

2.1.1 SELECTION CRITERIA

The time lapse between writing and publishing can vary from a few months for a journal article to 1 to 2 years for a full-length book. As a result, the publishing process

cannot keep pace with the influx of new ideas in an emerging field such as expert system development. Ideas that are new or current when documented may be obsolete by the time they reach the public. Accordingly, CSC restricted the literature search to those items published after 1984 and emphasized periodicals, conference proceedings, and technical reports to ensure a certain degree of currency.

Those books that were included were limited to those by authors widely recognized by the expert system community, such as Waterman and Hayes-Roth, or to those that seemed wholly relevant to the task. Those works were used primarily to develop the task's general theoretical background.

The research was greatly aided by a literature search conducted by the personnel of the Homer E. Newell Memorial Library at the Goddard Space Flight Center (GSFC). NASA library catalogs are stored on the NASA RECON data base developed by Lockheed. The bibliographic search strategy incorporates queries by selected keywords and combinations of keywords. The keywords used for this task included expert system, artificial intelligence, software engineering, software tools, computer systems design, and program verification. The result of the query was a 62-page bibliography with a synopsis of each entry. From this list, approximately 25 articles and books were selected for closer examination.

In addition, task members searched the Institute of Electronics and Electrical Engineers (IEEE) publications Expert, Computer, and Transactions of Software Engineering for relevant articles, as well as artificial intelligence (AI) conference proceedings and other publications. Selected articles often included references to additional

works, thereby further increasing the literature base. For any references published before 1985, the currency restriction was ignored if the information was directly applicable. One important work uncovered in this way was Expert Systems 1986 by Walker and Miller [29]. This work describes expert systems developed for diverse fields, including aerospace applications, and lists companies working in expert system development. This list proved to be a valuable source of potential interviewees.

2.1.2 LITERATURE DATA BASE

An early assessment of the volume of literature to be researched revealed the need for a data base to facilitate the cataloging of the reading material and the assessment of its usefulness to the study. The relational data base RBASE 5000, from Microrim, was chosen because it is easy to use and understand and has extensive, flexible report-generation capabilities. Appendixes A and B and the reference list were produced using this data base management system.

The expert system data base contains three tables for each book or article reviewed. The tables are linked by an identification number (LITID) thereby permitting reports to be generated containing data from all tables. The first table contains bibliographical information and includes fields for keywords and brief comments. To provide a contents summary of each book or article, a second table was set up. The characteristics of each individual paper read for this study were identified and assigned a weighted code, ranging from 1 to 5, indicative of the significance of the material relative to this study. Each paper was evaluated according to subject matter, i.e., quality assurance, configuration control, scheduling, or development life cycle in the technical area; and planning, organizing, monitoring,

and control in the management area. This TOPICS table, provided in Appendix B, made evident the dearth of information in the area of expert system management. The third table consists of an expanded synopsis of the book or article.

2.2 INTERVIEWS

Few of the more than 100 books and articles studied deal directly with expert system development methodologies (as per the definition of methodology given in Section 1). Most present personal experiences with the development of a particular expert system or portion thereof (e.g., knowledge acquisition only). Given the scarcity of literature on full-scale expert system development methodologies, interviews with individuals directly involved in expert system development became increasingly important and necessary.

The advantages of interviewing include the opportunity to

- Acquire the latest ideas and opinions on the subject, thereby avoiding the problem of "publication lag"
- Clarify statements, if needed, thereby avoiding misinterpretation, an important point in a field where terminology is not defined consistently
- Establish a rapport with those who may be useful when actually developing a methodology
- Acquire additional contacts

The disadvantages to interviewing are the extensive effort, time, and cost associated with interviews. Interviewing requires good communication skills; pre- and post-interview reviews and analyses; prearranged appointments; and, often, travel that is costly in terms of both time and money.

Interviewing resulted in a better understanding of the existing approaches and methods (or lack thereof) of expert system development. In addition, some interviews will be invaluable in the subsequent development of the expert system methodology. The interview results justify the efforts of the task members.

2.2.1 SELECTION CRITERIA

Originally, interview candidates included those with management as well as technical experience in expert system development, maintenance, and/or use. Preference was given to individuals involved in space applications located reasonably close to CSC/System Sciences Division (SSD). Software methodology experts and institutions well known for artificial intelligence/expert system research efforts (e.g., Carnegie Mellon University) were also included to acquire some broad points of view on the subject.

These sources, however, were not able to provide adequate information concerning the later stages of the expert system life cycle, especially the management and technical approaches to maintenance, quality assurance, and configuration control. Additional interviewees were sought who were directly involved in a system in operation for at least 2 years. CSC selected XCON, an expert system of Digital Equipment Corporation (DEC) that configures VAX computer systems. Unfortunately, due to the proprietary nature of XCON, CSC was unable to arrange an interview with DEC personnel despite several attempts. However, references [12], [41], and [73] provided some information on the maintenance of XCON.

2.2.2 SOURCES OF INTERVIEWEES

CSC uses a well-defined, structured methodology in both managerial and technical areas throughout the system life

cycle. Because CSC/SSD's expert system development efforts are in space applications, CSC personnel were among the first to be interviewed.

The Joint Artificial Intelligence Discussion (JAID) Conference, held for the CSC/AI community in May 1987, served as a valuable source of contacts. Task members reviewed a questionnaire distributed during the conference to identify individuals useful to the task objectives. CSC found that the responses on the written questionnaire were often ambiguous or incomplete, and occasionally different from responses given in later conversations. This experience prompted task members to abandon an initial idea of a mail survey (asking correspondents to fill out a questionnaire) and led to the use of personal and telephone interviews.

Another source of contacts with persons having artificial intelligence/expert system experience in space applications was the Second Conference on AI and Robotics, organized by GSFC and held in Greenbelt, MD, in May 1987. This conference led to contacts with personnel from the Hubble Space Telescope Science Institute (ST ScI) in Baltimore, MD, who had developed the Entry Processor System, an expert system that converts an astronomer-oriented description of a scientific observing program into a detailed description of the parameters needed for planning and scheduling.

Walker and Miller [29] list companies active in expert system development; this list was searched for those located in the Washington, DC, metropolitan area and involved with aerospace applications. Difficulties with the proprietary nature of commercial systems companies led to early abandonment of such contacts, with the exception of DEC.

Finally, Larry Hull, the task's Assistant Technical Representative, identified NASA personnel who lead artificial intelligence/expert system projects at different NASA centers.

The final list of interviewees is provided as Appendix C.

2.2.3 INTERVIEWING TECHNIQUES

Before conducting interviews, task members developed a high-level interviewing strategy. Task members decided to adopt a conversational form of interviewing, guided by but not limited to a base set of questions covering all areas of expert system development and maintenance. This set was reviewed and revised a number of times to include additional thoughts of task members as well as recommendations from trial interviewees. The final version of these questions is provided as Appendix D.

Each interview was tape recorded whenever possible to permit all task members to hear the actual interview and to facilitate analysis. When tape recording was not possible, the interviewer took handwritten notes. Each interview was carefully analyzed and conclusions drawn. In some cases, interviewees were recontacted, usually by telephone, to clarify points or to provide additional information.

The interviewing component of the research gathered information in those areas not adequately covered in the literature. In selecting candidates, CSC screened potential contacts by telephone. During screening, the interviewer briefly explained the purpose of the task and of the interview, and asked general questions to confirm the contact's ability to provide information useful to the task.

In the absence of well-defined and uniformly understood terminology in the expert system field, such a "preinterview" became especially important to prevent unnecessary traveling. Often, candidates who were not appropriate themselves identified others who were.

At the end of a successful preinterview, task members and the interviewees arranged a convenient time for a full

interview. At the start of each interview, the interviewer always asked permission to quote the interviewee. An attempt was always made to create an atmosphere of trust and confidentiality during the interview. After each interview, while impressions were still fresh (and especially after telephone interviews or when no tape recordings were made), the interviewer briefed other task members on the information acquired.

2.3 CONFERENCE PARTICIPATION

Task personnel attended various AI conferences and seminars, as well as a number of presentations held at NASA and CSC. This gave task members the opportunity not only to meet personnel who develop artificial intelligence/expert systems but also to acquire conference proceedings and other current written material. Conferences also proved to be a valuable mechanism for establishing contacts and obtaining names of individuals involved in related work.

The main criteria for conference selection were based on the following keywords: expert system, methodology, or space application. Some AI conferences sounded promising (as judged by their titles) until a review of their program showed them to be unrelated to the task's goals. For example, a methodology conference may have addressed a mathematical methodology in an expert system application but not a software engineering methodology for the building of the expert system. This again underscored the confusion over terminology in the field.

The following conferences/seminars were attended by task members.

The Software Rapid Prototyping seminar held in August 1987, in Washington, DC, addressed the conventional software development methodology, including the failure of the currently used waterfall method to accurately reflect some

features of the actual development process. This conference proposed an iterative approach (via a series of rapid prototypes). If valid, it narrows the gap between conventional software and expert systems development. See Section 3 for the detailed discussion of this subject.

The Third Annual Expert Systems in Government Conference, held in Washington, DC, in October 1987, was extremely useful in that two tutorials of immediate applicability were presented. Nancy Martin, an expert in expert system development methodology presented the tutorial "The Management of Expert System Development" and Barry Zack presented "Building Operational Expert Systems." The details of these are discussed in Section 3.

The Third Conference on Artificial Intelligence for Space Applications, held in Huntsville, AL, in November 1987, was useful in that topics directly relevant to the task were discussed. Furthermore, task members met K. Richardson who is a member of the Systems Autonomy Demonstration Project at NASA/Ames Research Center. His paper discussed an expert system life cycle developed by the group at NASA/Ames. Task members informally interviewed Mr. Richardson. The Ames life cycle is also discussed in Section 3.

Task members attended CSC and GSFC presentations on prototype expert systems. Troy Ames of GSFC discussed the knowledge-based system developed to support the data accounting and quality assurance functions of the Spacelab Output Processing System. CSC personnel demonstrated the final prototype of PRESS that was developed as part of a research effort. PRESS assists users in creating a constraint- and conflict-free schedule of activities. Both presentations addressed issues of current expert system development efforts that touched on the needs of an expert system development methodology.

SECTION 3 - ANALYSIS

This section describes the results of the study based on the literature search, interviews with experts, and conference attendance. Section 3.1 describes existing approaches to the development of expert systems uncovered during the study, while Section 3.2 compares the traditional software development methodology with those used to develop expert systems. Section 3.3 summarizes the results of the interviews, and Section 3.4 summarizes the results of the analysis. In this section, the term methodology is used often in the context of the references. It must be emphasized that this usage does not always correspond to that defined in Section 1.3. In the majority of the cases, the approaches to expert system development, called methodology in the references, represent a part of the methodology as defined in Section 1.3.

3.1 EXISTING APPROACHES TO EXPERT SYSTEM DEVELOPMENT

This section discusses the literature and conferences that address what is commonly thought of as a "methodology." The synopses of the papers and books read for this study (Appendix A) show that most literature addresses either a specific aspect of expert system development (e.g., requirements specification or verification and validation) or a tool or language applicable to the author's problem. From the literature, the following papers were identified as being the most comprehensive--Buchanan et al. [3], Waterman [1], Bobrow, Mittal, and Stefik [40], and Keller [5]. From the conferences, the seminar on software rapid prototyping [80], the tutorials by Zack [84] and Martin [85], and the work reported by Richardson and Wong [103] were the most comprehensive.

The literature may be divided into two categories. The standard expert system development approach is represented in the papers of Buchanan et al. [3], Waterman [1], and Bobrow, Mittal, and Stefik [40]. These are discussed jointly in Section 3.1.1. A structured approach taken by Keller and based on DeMarco [81] is discussed separately in Section 3.1.2.

The approaches of Zack, Martin, and Richardson and Wong were presented at conferences; these are discussed in Sections 3.1.3, 3.1.4, and 3.1.5, respectively. Finally, although not an expert system methodology per se, the conventional software development methodology, the waterfall method, has been included as Section 3.1.6 because it seems to be moving in a direction that may accommodate expert system development. The rapid prototyping seminar is discussed in the context of the waterfall method.

3.1.1 A STANDARD APPROACH TO EXPERT SYSTEM DEVELOPMENT

Buchanan et al. [3]; Waterman [1]; and Bobrow, Mittal, and Stefik [40] define stages or phases in the development of an expert system that are essentially the same, thereby permitting them to be discussed together.

Buchanan et al. [3] list identification, conceptualization, formalization, implementation, and testing as stages of knowledge acquisition. Waterman [1] uses this same terminology to describe the different development phases for building an expert system. Regardless of whether these terms refer to stages of knowledge acquisition or phases of development, they address the same activities. These activities are summarized by phase in the following description. Note that management issues are not discussed by Buchanan et al. or by Waterman.

Identification Phase--Identification begins the expert system development process. The knowledge engineer and the expert identify the problem, the necessary resources, and the goals of the expert system.

Conceptualization Phase--During conceptualization, the knowledge engineer and the expert meet frequently to more accurately define the key concepts and relations, and the control mechanisms. The knowledge engineer records these concepts and relations to make the conceptual basis for problem formalization permanent.

Formalization Phase--Formalization involves the expression of the concepts and relations defined in the previous phase by means of an expert system building language. The knowledge engineer needs to understand the nature and structure of the knowledge to be captured by the system, and must decide what expert system tool is best suited for the immediate application.

Implementation Phase--During implementation, the formalized knowledge is turned into a working program by the programmers. This is usually done in a prototype environment whereby various approaches are tried until the prototype expert system appears to perform in the same manner as the expert.

Testing Phase--The prototype is tested for its usefulness and performance. Each expert system requirement is verified and validated by a series of tests. This demonstrates that the knowledge representation is correct and that the inference engine reproduces the decision of the expert.

Testing may reveal that the system needs to be reformulated, redesigned, or refined. Reformulation entails changes in the identification and/or conceptualization phase, which in

turn affect the remaining phases. Redesign occurs in the formalization phase by changing the representation of the knowledge. Failure to meet performance requirements may also force a redesign. Refinement occurs via iterations throughout implementation and testing when relatively minor changes occur.

Iteration is a key element throughout the development process, especially for implementation and testing.

Iteration as a key factor in expert system development was supported consistently throughout the interviews. Waterman [1] emphasizes that for an expert system methodology to accurately represent expert system development needs, each phase must be able to interact with any other phase.

Partridge [53] characterizes AI program development as a run-understand-debug-edit cycle (RUDE cycle). Analysis of the process, especially in the context of the methodologies presented above, supports Partridge in his characterization of expert system development. Partridge warns that in its worst manifestation, the RUDE cycle is "hacking" whereas incremental analysis and redesign represent its better incarnation.

Bobrow, Mittal, and Stefik [40] list a similar cycle--identification, conceptualization, prototyping, creating user interfaces, testing and redefinition, and knowledge-based maintenance--as the stages of expert system development. The goals of their identification phase are the same as those already mentioned. The conceptualization stage encompasses both the conceptualization and formalization phases of Waterman and Buchanan et al.; prototyping is identical to implementation. Even though creating user interfaces is identified by Bobrow, Mittal, and Stefik as a distinct stage, the functions and activities

identified for this stage are essentially identical to those occurring in the implementation phase cited by Waterman and Buchanan et al. The activities of the testing and redefinition stage are likewise identical to those of Waterman and Buchanan et al. Only the knowledge-based maintenance stage is explicitly different from Waterman and Buchanan et al. Bobrow, Mittal, and Stefik state that during this phase a plan that provides for the testing, development, transfer, and maintenance of a large system must be made. The prototype has been accepted; it is now time to move on to the full-scale project.

Although Waterman; Buchanan et al.; and Bobrow, Mittal, and Stefik detail the activities that occur in the different phases of expert system development, their discussions remain too general for practical application. They do provide excellent overviews of the expert system development process; however, more detail is required to successfully apply their principles to a problem, as was discovered during the development of PRESS (comment by J. Bush and A. Critchfield during their interview).

3.1.2 A STRUCTURED APPROACH TO EXPERT SYSTEM DEVELOPMENT

Keller [5] describes a life cycle for an expert system that is based on the structured analysis techniques of Yourdon. He associates nine activities with the life cycle--survey, structured analysis, knowledge base design, design, systems integration, implementation, acceptance test, hardware analysis, and knowledge acquisition. Keller's list is deliberately nonchronological; several of the activities must occur in parallel. For example, knowledge acquisition and structured analysis, which Keller views as nearly identical, are done at the same time and by the same persons. He distinguishes the two by limiting knowledge

acquisition to the functional, logical content of the expert's domain, and structured analysis to the functional components of peripheral activities, such as the user interface. Structured analysis also includes the specification of the physical or technological components of the expert system.

Keller spends considerable time discussing early life-cycle activities, especially structured analysis. Applying structured methods, he provides a detailed example of a scheduling system, first by showing how the problem was selected and then explaining why the problem is suitable for implementation as an expert system. He does not, however, exhibit the same degree of thoroughness for the activities that follow analysis. Keller does include a discussion of the role of prototyping in expert system development but limits this to the use of PROLOG for this activity. The importance of Keller's book cannot be overlooked for those involved with the selection and specification of an expert system, although its weakness in the later stages of the life cycle is a shortcoming.

Table 3-1 maps Keller's life cycle into the standard life cycles discussed in Section 3.1.1. It must be emphasized that, although this table reveals an apparent degree of commonality among the various approaches, the activities within these phases are not necessarily the same. For example, the testing stage given by Buchanan et al. and Waterman refers to the testing of the original prototype, whereas that of Keller is broader and contains integration tests.

3.1.3 BUILDING AN OPERATIONAL EXPERT SYSTEM

At the IEEE Expert Systems in Government Conference, Zack [84] presented a tutorial on the construction of an expert

Table 3-1. Comparison of Three Approaches to Expert System Development

BUCHANAN ET AL. WATERMAN	BOBROW, MITTAL, AND STEFIK	KELLER
IDENTIFICATION	IDENTIFICATION	SURVEY
CONCEPTUALIZATION	CONCEPTUALIZATION	STRUCTURED ANALYSIS
		KNOWLEDGE ACQUISITION
		HARDWARE ANALYSIS
FORMALIZATION		DESIGN
		KNOWLEDGE ACQUISITION
		KNOWLEDGE BASE DESIGN
IMPLEMENTATION	PROTOTYPING	IMPLEMENTATION
	CREATING USER INTERFACES	
TESTING	TESTING AND REDEFINITION	ACCEPTANCE TEST
		SYSTEM INTEGRATION
	KNOWLEDGE BASE MAINTENANCE	

G632-1/12-87[6]

system. The presentation itself was on the same high level as Waterman [1] and Buchanan et al. [3], but it is useful in that it provides an additional perspective on the development of an expert system, especially in the management area.

Zack's model for development consists of application selection, system development, deployment, and maintenance and enhancement. The main functions of application selection are to identify a problem and then to determine both the value of and the feasibility of solving the problem with an expert system. Included in the feasibility issue is the commitment of management to implementing an expert system. In Zack's view, management not only must be committed to the expert system implementation but also must be willing to modify some of its organizational biases and accept incremental progress toward the problem solution.

System development begins with knowledge acquisition and moves rapidly into prototyping. Zack states that the objectives of prototyping can include a demonstration of technical feasibility for the entire expert system or a portion thereof, a demonstration of functional capabilities, and finally a successful prototype that proves the development team's ability to capture and utilize expertise. The prototype itself should focus on priority areas of the knowledge base and areas requiring either a new technology or a new problem-solving mechanism. The prototype results help define the development of the operational expert system. During prototype development, management must balance the iterative nature of prototyping with the needs to demonstrate progress and to arrive at a successfully developed expert system. Each prototype, although an incomplete system, must have specific objectives against which it is evaluated.

During system development, the functional specifications of the system are derived. Zack emphasized that the functional behavior of a knowledge-based system affects the complexity and the design of the knowledge base, and that the contents of the knowledge base cannot be specified in advance of its completion. Every expert system also requires an explanation facility, i.e., it must be capable of explaining its reasoning.

In the transition from a prototype development environment to an operational system development environment, Zack states that formal project management is instituted. The functional specifications and design are finalized and management prepares a detailed project plan.

Before the system is deployed, it must be tested. Both the knowledge base and the overall system require verification and validation. The knowledge base's conclusions and reasoning processes must be correct and its explanations lucid. For the overall system, the performance, usability, intelligibility, and acceptability must be evaluated.

Zack states that several issues must be addressed before delivery and deployment of the expert system. The user and the development team must agree on whether the deployed system will be used as developed or modified in response to the operational environment. For example, an expert system may be written on a large mainframe but ported to a smaller expert system shell. They must also agree on whether the delivered system will constitute a rewrite of the developed/prototyped system in a conventional language.

Zack emphasizes that less distinction exists between maintenance and enhancement of an expert system than in a conventional software system because a knowledge base is never really complete. Changes in the knowledge base can

occur because of discovered software errors, performance tuning, a changing operational environment, specification changes, new insight by the experts, generalization or simplification of knowledge, and new cases. To satisfactorily maintain an expert system, the expert as well as the knowledge engineer must continue to be available, and documentation must be complete and kept current.

Because of the high-level nature of the discussion, applying Zack's concepts directly would be difficult. In addition, any project wishing to implement these concepts must carefully analyze the timing of specific events. For example, a project manager may wish to develop a detailed project plan earlier than the time proposed by Zack. In addition, decisions about the delivered system (i.e., operational environment) should probably be made earlier than in Zack's deployment phase.

3.1.4 A COMPREHENSIVE EXPERT SYSTEM DEVELOPMENT METHODOLOGY

At the IEEE Expert Systems in Government Conference, Martin [85] presented the most comprehensive expert system methodology reviewed for this study. Her tutorial, "The Management of Expert System Development," described the cost and schedule drivers of expert system development, the composition of the development team, project planning, the acceptance criteria for the expert system, and the development stages and life cycle of an expert system.

Martin has developed the Expert System Controlled Iterative Enhancement (ESCIE) methodology. ESCIE has seven stages--initial feasibility study, rapid prototype demonstration, basic system usage, scope development system, refinement/enhancement, productization, and operations/maintenance.

During the initial feasibility study stage, the feasibility and advisability of working on a specific problem is analyzed and determined. The rapid prototype demonstration stage is used to illustrate the problem-solving capability of the expert system and to demonstrate the ability to rapidly obtain an executable system. The main purpose behind this demonstration stage is to garner the support of management, the user, and the expert. This stage can be performed in parallel with the next stages to investigate different knowledge-base designs or difficult aspects of the problem solution.

The basic system usage stage demonstrates that the expert system can perform the required reasoning and be beneficial to the users. The scope development system stage demonstrates the utility and the performance of the system over the scope of the desired functionality, while the refinement/enhancement stage is devoted to improving the system's performance, usability, capacity, and functionality. Ease of maintenance is emphasized during this stage. Within each of these last three stages, i.e., basic system usage, scope development system, and refine/enhancement, Martin places five phases--requirements analysis, specification-model, architectural design, design-implement-test, and evaluation. This scheme introduces iterativeness to ESCIE. Martin also relates these phases to those of Buchanan et al. [3] and Waterman [1] as shown in Table 3-2, which is an expansion of Table 3-1.

The productization stage is used to produce a marketable product, whereas the operations/maintenance stage provides for operations support, corrective action, and enhancements in response to changing environments.

Table 3-2. Comparison of Four Approaches to Expert System Development

BUCHANAN ET AL. WATERMAN	BOBROW, MITTAL, AND STEFIK	KELLER	MARTIN
IDENTIFICATION CONCEPTUALIZATION	IDENTIFICATION CONCEPTUALIZATION	SURVEY STRUCTURED ANALYSIS KNOWLEDGE ACQUISITION HARDWARE ANALYSIS	REQUIREMENTS ANALYSIS SPECIFICATION-MODEL
FORMALIZATION		DESIGN KNOWLEDGE ACQUISITION KNOWLEDGE BASE DESIGN	ARCHITECTURAL DESIGN
IMPLEMENTATION	PROTOTYPING CREATING USER INTERFACES	IMPLEMENTATION	DESIGN-IMPLEMENT-TEST
TESTING	TESTING AND REDEFINITION KNOWLEDGE BASE MAINTENANCE	ACCEPTANCE TEST SYSTEM INTEGRATION	EVALUATION

G632-3/12-87[6]

Martin describes each of her five phases as follows. During the requirements analysis phase, the user requirements are determined, acceptance criteria established, and the feasibility and advisability of proceeding with the project are evaluated. This description is identical to traditional software requirements analysis. The specification-model phase defines the problem solving required of the expert system. The architectural design phase determines the major components of the expert system, their structure, and their interfaces. Work is done not only on the structure of the knowledge base, but also on the inference engine itself. The goal of the design-implement-test phase is to obtain a working version of the system as early as possible so the user can validate it as it grows. Additionally, design decisions are implemented and evaluated immediately. As the design and code progress, the system is verified through unit and integration tests. Ultimately, the system is evaluated for its reasoning capabilities. smoothness of interfaces, visibility, ease of enhancement, performance, reliability, utility, cost-effectiveness, and scope.

Martin's methodology is an integrated approach, i.e., it integrates management and technical issues. ESCIE strives to provide the necessary management control that is often lacking in prototyping efforts and yet to maintain technical creativity. Martin defines estimation and computation approaches, cost/schedule drivers, and project roles. She provides job descriptions, descriptions for documents needed, and guidelines for time and manpower required for the rapid prototype through refinement stages for expert systems of different sizes.

Martin's approach, although comprehensive and detailed, cannot be readily adapted to the NASA/GSFC environment. The

ESCIE is geared to commercial ventures, i.e., to companies that wish to enter the expert system market and produce multiple copies of one product. For example, the main emphasis of the initial feasibility stage is to determine what markets are available for the company to pursue. Such an emphasis is not applicable in the NASA/GSFC environment where the emphasis during this phase would be determining which problems are suitable for an expert system solution. Similarly, the productization stage is superfluous in the NASA/GSFC environment as the expert system will not be marketed but rather will be used for a specific project or mission. Even if the productization stage is removed, the remaining stages require more detail, especially in the management areas of staffing, controlling, and planning. Her staffing profiles are also somewhat idealistic.

3.1.5 THE KBS LIFE CYCLE

Richardson and Wong [103] describe the knowledge-based system (KBS) life cycle they developed for the Systems Autonomy Demonstration Project at the NASA/Ames Research Center. They developed this life cycle during a workshop on the verification and validation of knowledge-based systems. In fact, the motivation for the KBS life cycle is to facilitate this verification and validation process.

The KBS life cycle consists of five phases: requirements, prototype, KBS build, test, and delivery and monitor. This life cycle is based on the traditional software life cycle and, although modified in places, it makes extensive use of the experiences gained with the traditional life cycle. The differences occur primarily in changes and improvements to the requirements document, the addition of an iterative prototyping loop, and the inclusion of the user in the development process.

The requirements document, as originally discussed in [103], refers to the high-level document that is given to the developers at the beginning of the project and before the serious task of requirements analysis begins. For traditional software, this document is meant to be complete; however, expert system requirements are not usually that well defined. Richardson and Wong state that this initial requirements document must accommodate the prototyping process by specifying the money, time, and other resources available for prototyping. In addition, the requirements document must include statements regarding the goals of the prototype(s) as well as the available resources. One goal of the prototype phase is to add details to the requirements document. During the prototype phase, the following activities take place: acquiring/extracting the knowledge, building the prototype, evaluating the results, augmenting the requirements document, and providing direction to the next prototype. The duration and extent of the prototyping effort are defined in the requirements document. Richardson and Wong recommend that a notebook be maintained during the prototyping effort to document design decisions. Although they do not specify whether all decisions are to be recorded, they do require that everything likely to be included in the final system be documented. The exact content of the notebook can be determined by the development team and, to a certain extent, is dictated by the system requirements.

Richardson and Wong define an increased role for the user during expert system development. They suggest that the user have more discretionary power in approving the use of the KBS. In addition, the users are represented by the domain expert during the prototyping phase. Also, because a KBS may be used in a different manner or in situations not

foreseen during the development phase, Richardson and Wong have extended the KBS life cycle to include user reaction and comment after delivery of the system. They also state that the user is least effective during the prototyping phase.

The KBS life cycle approach was discovered late in this study (early November). As a result, many details of the KBS life cycle are unavailable and cannot be fully evaluated. Interestingly, many of the concepts proposed by Richardson and Wong had already been identified as requirements for an expert system development methodology and had also been incorporated in the expert system development methodology outline being prepared for this task. Many similarities exist between the results and recommendations of this study and the KBS life cycle. These recommendations were presented in a preliminary version of this document prior to the receipt of the Richardson and Wong paper. These similarities may be due in part to the similarity between the NASA/Ames and NASA/GSFC environments and goals. They also may be due to the fact that the methodology outline being proposed by this task and the KBS life cycle are both based on the traditional software life cycle model.

Differences certainly exist between the methodology to be proposed to NASA/GSFC and the KBS life cycle. One immediate difference is the amount of user involvement. CSC research indicates the need for even greater user involvement than that indicated by Richardson and Wong. As more details of the KBS life cycle become available, other differences and similarities are sure to arise.

3.1.6 THE WATERFALL MODEL

Expert systems may or may not be standalone entities. They may be self-contained or they may be a part of a larger,

traditional system. Therefore, at a minimum, an expert system methodology will probably be required to adopt either directly or by reference many aspects of the traditional development methodology. Zack [84] emphasized that at least 50 percent, if not more, of the time spent on the development of an expert system involves the traditional activities of gathering data and coding it. Therefore, an expert system methodology may simply be a subset of a traditional methodology, and may need only to address those areas unique to expert system development and the interfaces between them and the traditional areas.

Only in the work of Richardson and Wong [103] was reference made to the use of the waterfall method for developing expert systems. However, task members concur that adopting existing aspects of the waterfall model, wherever applicable, may be desirable, especially given the wide familiarity with and overall acceptance of the waterfall model for traditional software systems.

This conclusion is based on a number of factors. First, most interpretations of the waterfall model recognize the importance of an integrated management and technical approach to software development, and place great emphasis on defining the activities that take place within the life-cycle phases, on planning and replanning these activities, and on monitoring and controlling the entire process. Even a preliminary assessment of the requirements to be levied on an expert system methodology indicates that both management and technical issues must be addressed.

Second, the life-cycle phases defined in the waterfall model are well defined and commonly used throughout most industries and disciplines, including space applications. Although individual methodologies may divide the phases

differently (e.g., establishing separate phases for preliminary and detailed design), there is, for the most part, agreement as to the basic phases required for good software development, the activities to be undertaken in each phase, and the end products of each phase.

3.1.6.1 Description of Waterfall Model

The waterfall model, as described here, is the basis for Mission Operations and Data Systems Directorate (MO&DSD) Systems Management Policy [45] and Software Development Policy [46]; for Defense Systems Software Development Standard (DOD 2167) [79]; and for CSC's Digital System Development Methodology (DSDM®)[1] [47], all successful system development methodologies.

The waterfall model is divided into six distinct phases, defined as follows:

Requirements Analysis--Involves the analysis and definition of the complete set of functional, performance, interface, data base, and user-system requirements. These are documented in a requirements specification, which serves as input to the next phase, and are presented to the client in a formal requirements review.

Design--Involves two design stages. Preliminary design involves the allocation of requirements defined in the previous phase to high-level components. These are documented in a preliminary design specification and are presented to the client in a formal design review. Detailed design involves a more detailed definition of the design by further breaking down the preliminary design, allocating the

[1] DSDM is a registered trademark of CSC.

high-level functions to individual units. Again, these are documented and presented formally to the client.

Implementation--Comprises the design (i.e., program design language), coding, and testing of each unit defined in the detailed design. For large software development efforts, the build approach is taken during this phase. Builds permit subsets of requirements to be met in each build, thereby avoiding the "big-bang" approach, i.e., integrating the system initially in one large step. Each build is documented in an updated design document and, depending on the system, a user's guide, operations manual, or maintenance manual. Each build begins with a review that is usually somewhat less formal than a critical design review.

Integration and Test--Involves the integration of the individual software elements and the testing of this integrated entity as a whole. Requirements, design, and operations/user documentation serve as the basis for test documentation. This phase ends with a test readiness review that documents the test results and confirms that the system is ready to proceed to the next phase.

Acceptance Test--Involves verification by an independent group that the system as developed meets the requirements as specified in the requirements specification, and that the operations/user documentation accurately represents the system. An operational readiness review or acceptance review concludes this phase. After the final review, the developed system is either delivered to the client or enters the maintenance phase.

Maintenance--Includes the activities performed in all of the above phases. The extent of these activities depends on the nature and scope of changes introduced or of problems encountered.

This life cycle is illustrated in Figure 3-1 and is taken from reference [46]. It represents the methodology used by the MO&DSD at GSFC.

3.1.6.2 General Limitations of the Waterfall Model

The waterfall model is a product of the 1970s and has been criticized recently as not being wholly applicable to or responsive to today's software development environment. Specific criticisms of the waterfall method appear to focus on the fallacy of phase independence, the long duration of the life cycle, and the lack of early user involvement.

The waterfall method assumes that each phase of the life cycle is completed before the next is begun. In fact, the output from one phase serves as input to the next. As Lantz states in [68], the system development process may be viewed "as a set of rigorously distinct phases performed at separate times." Lubars and Harandi [50] point out this is not actually true since feedback takes place between the implementation and the specification of requirements. New requirements commonly are added or deleted after the "completion" of the software requirements review. In CSC's experience, even testing, the last phase, can have an impact on the implementation phase as well as on the earlier phases. In reality, no phase is ever completely finished. Lubars and Harandi [50] also indicate that the time between the introduction of specifications (requirements) and the completion of the final product is often so long that specifications often change. If specifications are frozen, the product may well be obsolete upon delivery. Gladden [69] points out that this long time interval can also erode the customer's confidence in the work being performed, which may in turn be manifested in new, altered, or expanded requirements.

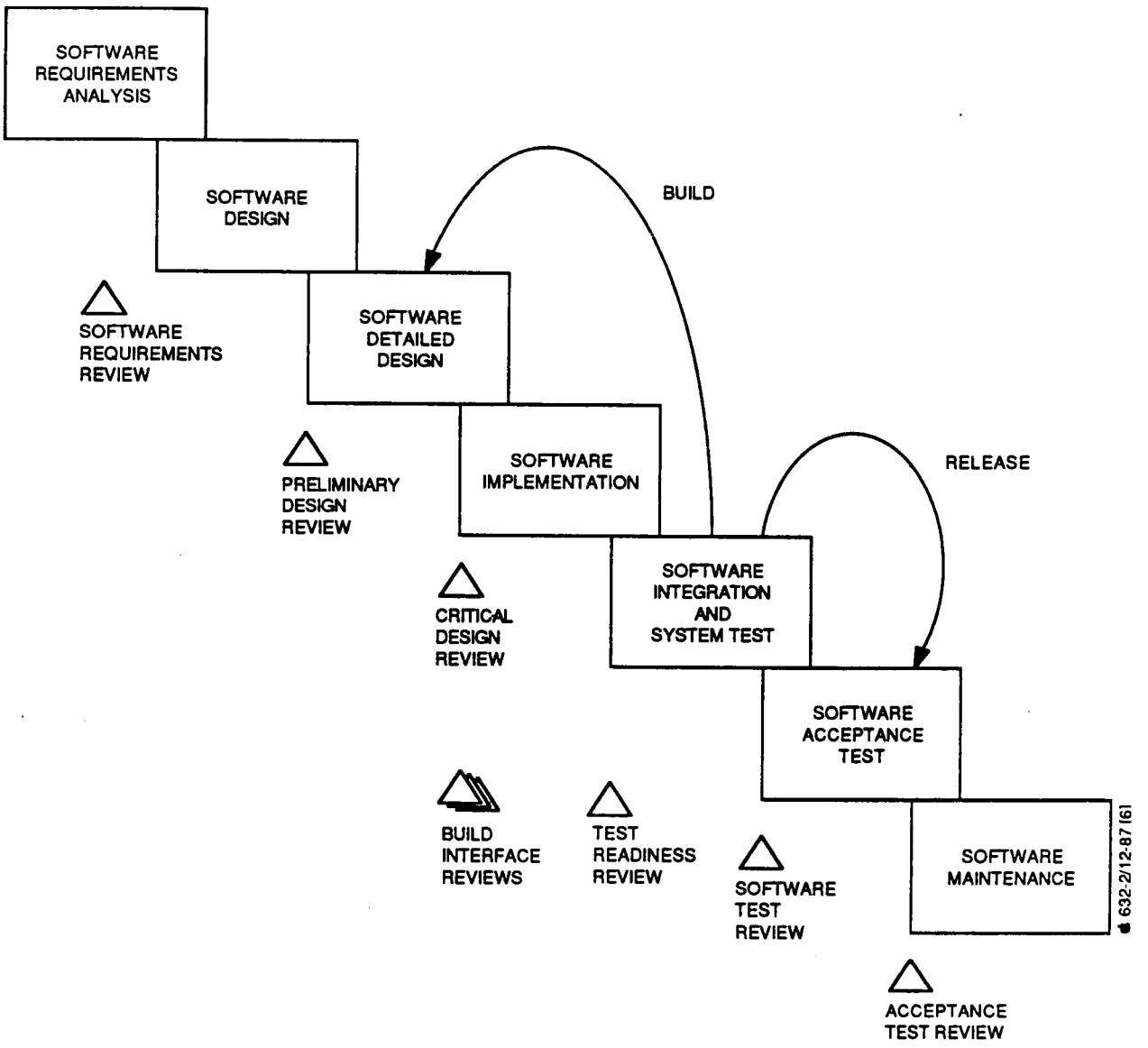

Figure 3-1. The Software Life Cycle (adapted from [46])

Lubars and Harandi [50] also criticize the lack of early end-user involvement that can lead to a system that does not address user needs and hence, which the users will not or cannot use.

One way to mitigate risks associated with the long duration and the lack of early user involvement has been to develop the system in a series of builds. A build can be either demonstrated or released to the customer for interim use during development. The customer can monitor the status of the development effort, and the users can exercise the system to determine whether their requirements are being properly implemented. Demonstration of the build assures the customer that the requirements are being properly addressed, but unless the end user is included in this process, the user may still disagree with this assessment. The customer and user often view the product from different perspectives.

Gladden [69] also claims that the "villain" in any software failure is a set of incomplete, poorly thought-out requirements. This is not a limitation inherent in the methodology but is rather an improper application of the methodology. The methodology "allows" changes but only in terms of a change in the scope of the development effort. Extensive late changes can have serious consequences to the development project.

Agresti [77] has presented an interesting limitation to the waterfall methodology. He points out that tools exist for all phases of the life cycle, but that in many cases, the tools and the software development environment span several of the phases of the life cycle. He contends that tools that span multiple phases challenge the wisdom of the rigid partitioning of software development into phases.

An additional limitation to the waterfall method is that it is document driven. Each phase produces a product, usually a document or a set of documents, that "completes" that particular phase. Due to technological advances or to reinterpretation, changes to requirements do occur. When this happens, the effect of the change can ripple throughout the entire life cycle, often affecting design, code, and product description (specification). There really is no such thing as a "simple requirement change" and numerous changes can lead to extensive work. This is one reason researchers are investigating automated code generation tools and specification languages (cf. [22], [25], [50], [70], [74], [88]).

Recently the Defense Science Board issued its Final Report on the Software Task Force [78]. In this report, DOD 2167 is criticized for being too rigid, not allowing prototyping, and failing to conform to "best modern practices." Some of this criticism was addressed in the MO&DSD methodology, which does allow prototyping, in certain cases, up to the point of the critical design review.

John L. Connell proposed a new methodology at a formal 2-day seminar on software rapid prototyping in Washington, DC, sponsored by the Education Foundation of the Data Processing Management Association. The intent of Connell's methodology is to free the traditional life cycle from its rigidity and phase independence and to involve the user early in the life cycle.

Connell suggests that a more appropriate name for this methodology is dynamic requirements modeling. He feels the term rapid prototyping conjures up many negative illusions, including a license to hack. Connell's approach combines the iterative nature of prototyping with the structured

techniques of DeMarco [81]. His rapid prototyping environment consists of a relational data base, suitable hardware, and structured software development techniques [80]. His approach does not have universal application. It is geared to highly interactive systems and may not be applicable to scientific problems typical of space applications. Nevertheless, several concepts presented by Connell were assessed as useful and are included in the recommendations presented in Section 4.

To summarize, the limitations of the waterfall method are phase independence, long duration, and lack of early user involvement. The major limitation is that it does not allow iteration between the phases of the life cycle. The build approach satisfactorily addresses the long-duration and user-involvement problems but fails to deal with the issue of phase independence. It permits iteration within a phase but does not allow another iteration of requirements analysis or system design after implementation has begun unless requirements change dramatically.

3.2 COMPARISON OF WATERFALL AND EXPERT SYSTEM APPROACHES

A careful analysis revealed that many analogous activities take place within the traditional waterfall method and the general categories of development of an expert system detailed in Section 3.1.1. In this section, the expert system approaches are viewed at in the context of the waterfall method, with similarities and differences identified and discussed. This comparison is organized along the lines of the traditional life-cycle, beginning with requirements definition and ending with operations and maintenance, since there is common agreement as to this life cycle.

Table 3-3 summarizes the results of the comparison of the waterfall method with the standard expert system development

Table 3-3. Comparison of Traditional and Expert System Software Life Cycles

TRADITIONAL SOFTWARE SYSTEM	EXPERT SYSTEM
REQUIREMENTS ANALYSIS	KNOWLEDGE ACQUISITION
REVIEW REQUIREMENTS FOR COMPLETENESS	GATHER KNOWLEDGE
DEFINE ADDITIONAL REQUIREMENTS REMOVE REDUNDANT/INACCURATE REQUIREMENTS	REFINE KNOWLEDGE
SOFTWARE REQUIREMENTS REVIEW	COMMUNICATION BETWEEN EXPERT AND KNOWLEDGE ENGINEER
DESIGN	KNOWLEDGE REPRESENTATION
TOP-DOWN STRUCTURED DESIGN	KNOWLEDGE BASE CONTAINS FACTS AND RULES KNOWLEDGE REPRESENTED THROUGH RULES, FRAMES, SEMANTIC NETS
DESIGN REVIEWS	COMMUNICATION BETWEEN EXPERTS, KNOWLEDGE ENGINEER, PROGRAMMERS
IMPLEMENTATION	IMPLEMENTATION
CODE SYSTEM USING PROGRAMMING LANGUAGE	"CODE" SYSTEM USING AI TOOLS
UNIT TEST	"UNIT" TEST
BUILD APPROACH – INCREMENTAL IMPLEMENTATION OF REQUIREMENTS	ITERATE OVER KNOWLEDGE ACQUISITION, AND REPRESENTATION, REQUIREMENTS ANALYSIS, DESIGN, AND CODING
SUCCESS = ACCEPTANCE OF THE SYSTEM	SUCCESS = KNOWLEDGE GAINED REGARDING KNOWLEDGE REPRESENTATION AND DESIGN
VERIFICATION AND VALIDATION	VERIFICATION AND VALIDATION
UNIT TESTS MODULE TESTS INTEGRATION TESTS INTERFACE TESTS ACCEPTANCE TESTS	UNIT TESTS MODULE TESTS INTEGRATION TESTS INTERFACE TESTS ACCEPTANCE TESTS
OPERATIONS AND MAINTENANCE	OPERATIONS AND MAINTENANCE
CORRECT ERRORS ENHANCE SYSTEM	CORRECT ERRORS ENHANCE SYSTEM, INCLUDING CHANGING KNOWLEDGE
CONFIGURATION MANAGEMENT – SOFTWARE – DOCUMENTATION	CONFIGURATION MANAGEMENT – SOFTWARE – DOCUMENTATION – KNOWLEDGE BASE

G632-4/12-87[6]

methodology. The table lists only the major activities within the traditional software methodology and compares them, as much as possible, to analogous or parallel activities. Knowledge acquisition and knowledge representation are compared to requirements analysis and design, respectively. This does not imply that knowledge acquisition is the same as requirements analysis or that knowledge representation is design. Elements of requirements analysis and design are found in knowledge acquisition and representation, but the latter are much broader concepts, as discussed in Sections 3.2.1 and 3.2.2. Also, for the traditional waterfall software development, the stages follow sequentially; for the expert system, the ordering in the table is not chronological. Knowledge acquisition, knowledge representation, and implementation may occur in parallel.

3.2.1 REQUIREMENTS ANALYSIS PHASE

In the traditional requirements analysis phase, a set of requirements drives system development. The development team analyzes the initial set of requirements for completeness; defines additional requirements from the initial set; and removes redundant and/or inaccurate requirements with management concurrence. This phase ends with a formal requirements review at which time a baseline is established and, theoretically, the requirements are finalized. In practice, requirements change and the baseline is updated through configuration control mechanisms.

An analogous process takes place during expert system development. The development team generally receives a description of the desired results or of the purpose of the expert system, rather than a set of requirements (i.e., no requirements, per se, are provided). The development team

performs knowledge acquisition rather than requirements analysis, knowledge being the essential part, i.e., the driver, of the expert system. That knowledge acquisition is a key concern in expert system development is evident from the large number of papers written on the subject (see, for example, [28], [33], [34], [42], [64], [65], [76], [101]).

Most often, development of an expert system requires that an expert be available who possesses a narrow and deep domain of expertise. This is not usually true of space applications; more likely, a body of experts will collectively supply the development team with the required knowledge. The development of space applications expert systems, therefore, introduces an additional problem of selecting from among possibly conflicting knowledge. The issue of multiple experts has been discussed by Boose [19] and Woolf and Cunningham [24]. Experts must reach a consensus before an expert system can be implemented. Therefore, a mechanism (e.g., meetings) to arrive at a commonly agreed to and acceptable solution is needed. Zack [84] and Martin [85] both propose that a lead expert be appointed for the project.

The existence of expert systems introduces a new philosophy to problem solution. Traditionally, a problem is analyzed and, if feasible, high-level requirements are defined. Then the development team begins the software development effort with an analysis of the given requirements. Now the problem must first be analyzed for its suitability as an expert system, i.e., does an expert system represent the correct approach to the problem solution. This analysis must occur in the concept definition phase and is performed before any requirements are given to the development team.

The quality of traditional requirements definition is judged by the completeness of the requirements at the time of the

requirements review. The goal of the development team is to have everything defined in the initial phases to the greatest extent possible. The opposite is true for an expert system. Typically only expert system goals and high-level requirements are known at the beginning. The requirements are not thoroughly defined until the completion of the final prototype.

Who conducts the various activities related to requirements analysis? In the traditional method, the development team during the requirements analysis phase consists of experienced system engineers and/or senior analysts. For an expert system, the lead figures in the early stages are a knowledge engineer (who may also be the developer), who is skilled in capturing the knowledge of the expert, and the expert, someone capable of conveying expertise to the knowledge engineer. These may not necessarily be separate individuals in the NASA environment, where many of the experts possess enough computer knowledge to encapsulate their expertise. Jackson [52] describes his experiences as both the "expert" and the developer of two small expert systems used to evaluate scientific proposals for observations with the Hubble Space Telescope. In an interview, he stated that writing his systems in OPS5 was rather straightforward even with his limited computer background. A computer expert familiar with OPS5 and AI was available to help when Jackson did experience some difficulties. This single-person approach is not practical for large systems when more than one knowledge engineer is necessary, but it does suggest that an expert, with the desire and ability, can take a more active role in expert system development over and above simply providing knowledge and expertise.

3.2.2 DESIGN PHASE

In the traditional methodology, the requirements baselined at the requirements review serve as the system's goal. The next phase is to design a system that, when implemented, will satisfy these requirements. During this phase the design is depicted graphically, often in the form of structure charts. The design is reviewed at formal reviews.

For an expert system, a similar graphical procedure may be used but the design philosophy differs. In the traditional methodology, the designer works from a specific set of requirements; whereas for an expert system, the developer formalizes the structure of the expert system, while trying to determine through knowledge acquisition the actual functionality and behavior of the expert system. The knowledge engineer is still acquiring knowledge from the expert while defining the representation of that knowledge. Thus knowledge acquisition and knowledge representation are concurrent and equally important activities in this phase. A useful discussion of the design of expert systems is given in detail in [82] and summarized in [100], where Kline and Dolins identify five questions that must be considered when designing an expert system:

- When is the information needed to solve the problem available?
- What kind of connection is there between the evidence and the hypotheses?
- What counts as a solution and are multiple solutions allowed? If so, how many solutions are there likely to be?
- How accommodating is the expert system program environment?
- Will the program expend its effort wisely?

One aspect of traditional and expert software design is the modularization of the software structure. It is considered important to structure the software into independent modules so that changes affect as little of the software as possible. The guiding principle behind this premise is ease of maintenance. This philosophy is also being applied in expert system development. Lindenmayer, Vick, and Rosenthal [9], Medeiros [37], and Jacob and Froscher [44] discuss the importance, in terms of maintenance, of modularizing expert system software. Their views were reinforced in interviews with Lindenmayer, Jon Buser, and Andrea Weiss who all strongly advocated a modularized design. Unfortunately, no methodology touches on how to modularize an expert system. For PRESS, Bush and Critchfield combined related rules into a module that was independent of all other modules; rules that were used by more than one module were placed in a utility module. Buser suggested that the information hiding techniques of Ada may be useful for modularization. Other possible ways to modularize an expert system are to separate the inference engine from the knowledge domain, to incorporate modularity into the knowledge base (as suggested by Bush and Critchfield and by Buser), and within the knowledge base separate the rapidly changing knowledge from the more stable knowledge.

It is generally recognized that modular structure in traditional software has made software design and maintenance easier and has helped control cost. Keller [5] extends the techniques of structured analysis and structured design to expert system development. Whether this is the proper approach to achieving expert system modularity remains to be seen. Ebrahimi [67] suggests another approach. He develops high-level protocol templates for the interaction of the domain expert, the knowledge engineer,

and the system during knowledge acquisition; identifies areas that are likely to be implemented as independent modules with predefined interfaces; and analyzes the acquisition, reasoning, and explanation subsystems for user input, the transformation of the user input to subsystem input, subsystem input, subsystem functions, subsystem output, transformation of the subsystem output to user output, and the user output.

3.2.3 IMPLEMENTATION PHASE

Traditional development differs most from an expert system development during implementation. In the traditional approach, the programmer takes the design, which satisfies the requirements, and converts it into a working program using the appropriate language. The expert system developer, however, has an advantage in that the available expert system development tools are quite advanced. For example, shells make the "coding" of an expert system much easier than that of a traditional language (e.g., FORTRAN) since shells are often closer to English and have the inference engine built into them. This allows the developers to concentrate on the implementation of the expertise rather than on coding low-level details.

The role of the developer is also different. In the traditional methodology, the programmer is relatively constrained (i.e., is allowed to implement only the design provided based on the requirements provided). In theory, the programmer is not invited to investigate the requirements or the design for flaws or deficiencies, as these are expected to have been exposed in earlier reviews. The developer of an expert system, on the other hand, is still exploring ways to improve the system, extend its capabilities, and determine its completeness. The developer

is still capturing expertise and incorporating it into the framework of the expert system even as the expert system itself is being implemented.

The iterativeness of expert system development can be compared somewhat to the build approach in traditional software development. In the build process, the requirements of the total system are divided into subsets so they can be implemented incrementally. A specific requirement may be partially implemented in one build and completed in a later build, or it may be ignored until a later build. The last build, however, must totally implement all the requirements. An expert system can be implemented in a similar fashion. An idea can be tried and tested in an early version of the system and expanded in later versions if proven successful.

The build approach and the prototype iterations of an expert system differ in a number of ways. First, the build approach takes the known, and presumed complete, requirements and divides them among the builds. In an expert system, all requirements are not known initially. Hence, iteration occurs within and between the analysis, design, and implementation phases, not just within the implementation phase.

Another difference, and a major one, is the perception of success and failure. In expert system development, a prototype may be considered successful even if it fails. The expert system developer accepts the possibility that an idea may have to be discarded if it does not work, even if this implies a complete restart. It is, in fact, expected that some paths tried in an expert system will not work. Many even recommend that the tool or shell originally purchased to accomplish the implementation be discarded if

it is not satisfactory. Such a possibility must be expected and requires a carefully prepared contingency plan. On the surface, this could be construed as "starting over from scratch" when in reality progress has been made by eliminating one or more design or implementation options.

In the build approach, success is measured with the acceptance of the implemented software by the independent test team; i.e., the software as built meets the requirements written some time ago. Software that does not fully or accurately meet requirements has discrepancy reports written against it; rarely is it discarded wholesale. Such a move would be unplanned and would have major schedule and budget repercussions.

3.2.4 TEST PHASE

The purpose of testing is essentially identical for traditional software systems and expert systems: to ensure that the functional and performance requirements are satisfied. In general, this view is supported by the interviews, although dissenting opinions [66] do exist. An assumption made in determining that this phase is the same for both types of software is that when testing begins, the requirements of the expert system have been established and are documented. Only then can verification and validation proceed. Green and Keyes [66] do not accept the validity of this assumption. They state that requirements specifications for expert system software are often nonexistent, imprecise, or rapidly changing. Often they are obtained from a system specification (Type A specification) or from informal specification by prototyping and refinement. The intermediate specifications (Type B5 or C5) are either not produced or, if produced, not adequate. The result is that the final requirements cannot be traced back

to the original specifications. If precise requirements are not available, then it is not possible to write the precise test procedures required of conventional verification. Even if traceability to the requirements were possible, Green and Keyes believe that conventional verification is still not possible because the desired results are the product of the interaction of the knowledge base and the inference engine. Finally, Green and Keyes state that the results of the expert system cannot be checked objectively; that is, the results are compared to the results given by a human expert. The human, of course, is subject to prejudices and biases and can even be in error. Accordingly, a vicious cycle occurs: no one requires the verification and validation of an expert system; verification and validation are not required because no one knows how to do it; no one knows how to do the verification and validation of an expert system because no one has done it.

A more optimistic point of view is given by Culbert, Riley, and Snavely [95]. They state that it is possible to verify and validate expert systems if the differences between traditional software and expert systems are recognized and if expert systems are written to make verification and validation easier. Requirements can be documented after the completion of the prototyping phase. This allows the test plan, from which the test procedures are derived, to be written. They also suggest that for the NASA environment, Flight Technique Panels can be used to verify expert systems. These panels regularly review not only the procedures used to resolve the problem but also the analysis techniques used to develop the procedures. Validation occurs through exhaustive simulations.

Guidelines for expert system verification are given by Goodwin and Robertson [91], and guidelines for expert system validation are given by O'Keefe, Balci, and Smith [98]. Castore [94] presents a formal approach to verification and validation of knowledge-based control systems. Goodwin and Robertson provide the definition of a verified expert system, discuss verification of traditional software, and conclude that the verification techniques of traditional software can be extended to include expert system software. More attention needs to be given to design methodologies and documentation and coding standards. Once these are established, tools can be developed to facilitate the verification of an expert system.

O'Keefe, Balci, and Smith [98] discuss problems that may occur when validating an expert system. Specifically, they discuss what to validate, what to validate against, what to validate with, when to validate, how to control the cost of validation, how to control bias, and how to cope with multiple results. Their guidelines for validating expert system performance can be summarized as follows:

- Validate the expert system only against an acceptable performance range for a prescribed input domain.
- Build validation into the development cycle.
- Consider the risk in using invalid systems (user's risk) relative to the risks in not using valid systems (builder's risk).
- Choose the appropriate qualitative validation method.
- Use quantitative validation methods where applicable.

Gaschnig et al. [14] give a thorough, comprehensive, high-level perspective of expert system testing. They discuss what to test in an expert system and when to test it, and they suggest four principles that should be applied when evaluating expert systems:

- Complex objects or processes cannot be evaluated by a single criterion.
- The larger the number of distinct criteria evaluated or measurements taken, the more information will be available on which to base an overall evaluation.
- People will disagree about the relative significance of various criteria according to their respective interests.
- Anything can be measured experimentally as long as the mechanism for taking the measurements is defined.

These principles apply equally to testing traditional software. Later, Gaschnig et al. give seven key issues that must be addressed in evaluating expert systems:

- The need for an objective standard of excellence
- Concerns regarding biasing and blinding
- The elimination of irrelevant variables
- The definition of realistic standards of performance
- The need for sensitivity analysis
- Problems with confounding interactions among knowledge sources
- The need for realistic time demands on evaluators

Again, these issues are equally appliable to testing traditional software.

3.2.5 OPERATIONS AND MAINTENANCE PHASE

After the traditional system has been accepted and made operational, the system enters the maintenance phase. In this phase, problems not discovered during development and testing are identified and corrected, and enhancements are made under carefully controlled conditions. The situation is similar for an expert system, with one important difference: the maintenance of the knowledge base.

Frail and Freedman [51] present some observations and guidelines for the delivery of an expert system:

- Delivery of an expert system should be considered in the earliest stage of development.
- Production rules are useful for prototyping a small system rapidly to demonstrate feasibility.
- Adding rules does not necessarily improve an expert system.
- If an expert system product has been successfully used to solve problems in several separate but similar domains, it may be cost-effective to transform that system into a shell.
- Is the final deliverable system still an expert system?

Not all of these observations and guidelines apply specifically to the delivery of an expert system, but they do raise some interesting philosophical points that need to be addressed during the maintenance phase.

Ther first observation applies to both expert and traditional systems; the second is more concerned with design. In fact, Frail states that after the demonstration of feasibility, the remainder of the system should be developed with static knowledge. No additional rules or knowledge should be added to the system.

The third observation has significant maintenance implications. The major difference between an expert system and a traditional system is that the expert system must be concerned with truth maintenance. When a change is made to the knowledge base, has the integrity of the knowledge base been compromised? In the traditional software system, the integrity of data in the data base is the responsiblity of a data base administrator. An analogous position may be needed on an expert system project to control the knowledge base.

The fourth observation has to do with economics for the company responsible for developing the expert system, whereas the final observation is a philosophical one. The question is whether a delivered expert system remains stagnant. Knowledge continually increases as a person learns new facts or better ways to do something. Will the delivered system retain the same knowledge base, or will the knowledge base change in parallel with human knowledge? If the latter, the maintenance of an expert system must include mechanisms for changing the knowledge base.

An important activity during operations and maintenance is configuration control of the system and its documentation. In addition, whenever a change is made to the system, regression testing is essential. A key to the ease of maintenance is the modularization of the system, as was discussed in Section 3.2.2. In the seminar on software rapid prototyping, Connell stated that configuration control of a prototyped system should be the same as that of a traditionally developed system. When a change has been identified, however, it should be implemented using prototype or iterative paths rather than the rigid, traditional path.

Practical experience has been gained with the maintenance of R1 or XCON, a large system used by DEC to configure hardware items into a compatible system. Kraft [12], Soloway, Bachant, and Jensen [41], and van de Brug, Bachant, and McDermott [73] describe the difficulties of maintaining this system. Two main difficulties have arisen in the maintenance of XCON: as XCON grows, it is becoming more heterogeneous, predictability in the rule base is exceedingly difficult. The maintainer needs to know more information than is contained in the knowledge base in order to do the job. The reason a specific rule exists or why rules fire in a specific sequence is not known. In addition, it is not obvious what effect changing a rule has on the remainder of the knowledge base. The maintainer needs to understand at least a major portion of the knowledge base before a rule can safely be changed. The need for structure and adequate documentation is obvious. Soloway et al. refer to a programming tool that has been developed that "provides on-line enforcement of coding guidelines" that could mitigate the documentation problem.

3.2.6 MANAGEMENT CONTROLS

CSC, as well as other companies, has learned that successful delivery of a quality product requires certain controls. These generally are implemented through the product assurance and project control elements of the project.

In the readings, these areas were essentially ignored, even though authors agreed to the necessity of addressing them. The interviewees confirmed this position but had no idea how to define or implement these areas; they were interested more in "technical issues."

Project control monitors adherence to budgets and schedules, reporting results regularly to permit corrective action at the earliest possible time if variances are found. Product

assurance monitors the technical quality of the product through a series of reviews, inspections, and audits, the results of which are reported to management. Management thus has the data needed to retain control over the project. An expert system development methodology clearly must address these issues, because the nature of expert system development makes monitoring and control particularly difficult.

3.3 INTERVIEW RESULTS

As indicated in Section 2.2, the interview process, although time consuming, supplied useful results. The interviews allowed task members to follow up immediately on specific issues that arose and to ask for more detail about a specific issue that would not normally be written in a paper. The list of interviewees is given in Appendix C.

Analysis of the interviews uncovered a number of common themes. Interviewees often had similar responses to or suggestions about various issues or areas of concern. This section is organized by the major topics raised during the interviews. Responses in each area are synthesized and summarized rather than detailed individually. This approach eliminates redundancy and highlights those comments considered to be most significant or universal. In some instances, the source of a comment is placed in parentheses following the comment.

Methodology--Although most interviewees did not follow a formal methodology, all agreed that it is important to develop methodology that assists in building a better, maintainable expert system.

Communication--Informal and frequent communication with the expert and users is important for the developers. The lack of contact with an expert negatively affected a prototype development (Das); on the other hand, satisfactory results

were achieved with adequate access to the expert and/or users during the development of an expert system (Gilstrap, Botten, and others).

Frequent communication can be achieved by using a prototyping technique (all interviewees). The prototype allows the users and experts to comment timely on the approach taken by the developers and to clarify any misunderstandings. Artificial intelligence tools encourage prototyping and thus facilitate communication (Botten, Bush, Critchfield, Das, and Gilstrap). In addition, the main users and the expert must be part of the project (Weiss), but the expert must be willing to give time to the project (Botten, Bush, Critchfield, Das, and Gilstrap).

Tools--Tools are so important to expert system development that their development must be made part of the expert system development effort (Buser and Knoblock). Tool development must be planned, budgeted, and scheduled; accordingly, at least one project member should be assigned full time (at least initially) to that task (Buser).

Scheduling--All interviewees acknowledge that the lack of precise requirements leads to difficulty scheduling expert system development. The experience level of the knowledge engineer should help to minimize this problem: the more experience the knowledge engineer has with expert systems, the more accurate an estimate of the scope of the project and, hence, the more accurate a schedule for the expert system development effort (Gilstrap).

Implementation--It may be necessary to change the implementation language to satisfy the system's performance and operational requirements. However, the use of artificial intelligence tools is advantageous in the beginning of the project as a design environment for expert systems. Although such an approach appears to involve more

effort, it results in the creation of a better product that does not require immediate modification upon delivery (Botten, Bush, Critchfield, Gilstrap, Jackson, Lindenmayer, and Miller).

Verification and Validation--All interviewees acknowledge the difficulty of thoroughly verifying and validating an expert system. No standard approach currently exists, even though the need for one is universally recognized. Das created scenarios that identify and test the system's behavior in different situations. Bush and Critchfield created a "system behavior" matrix to help generate test cases for major functional combinations. All agree that the expert is an important participant in the verification and validation of an expert system.

Maintenance--All interviewees state that it is (or will be) difficult to maintain an expert system. The interviewees suggest that the following should make maintenance easier:

- Use a structured approach to construct the knowledge base by
  - Separating rapidly changing knowledge (data-like knowledge) from slower changing knowledge (rules-like knowledge)
  - Separating the inference engine from the knowledge domain
  - Separating the knowledge base by function, i.e., incorporate modularity into the knowledge base
  - Maintaining a rule dependency chart (Das and Lindenmayer)
- Create knowledge acquisition tools that help generate rules in a standard style and format (Knoblock and Weiss).

- First implement changes using an artificial intelligence tool; then, after clarifying all concepts, incorporate the changes into the operational system (Gilstrap and Botten).

3.4 SUMMARY

The results of the analysis show that no formal, documented, comprehensive methodology exists for the development of an expert system, with the exception of Martin's ESCIE. The lack of a formal methodology contributes to the current state of affairs, wherein each development project must essentially learn from its mistakes. The need for a formal methodology is obvious.

The analysis also revealed enough similarities between the activities in traditional software development and those in expert system development to make it feasible--and practical--to develop an expert system development methodology within the framework of the waterfall model. The movement of the waterfall model toward a prototyping environment further supports this contention. Accommodations must be made for several important aspects of expert system development as follows:

- Allow iteration within a phase and across phases
- Permit requirements to be derived in parallel with system development
- Redefine the success criteria for intermediate prototypes
- Increase the scope of user involvement
- Identify a means of measuring progress (This is particularly difficult because no objective measure yet exists to measure progress in a prototyping environment.)

- Broaden lines of communication between the development team and the users; frequent communication is critical between the expert and knowledge engineer and among all project members during the prototyping effort
- Develop new control mechanisms, especially in the prototyping environment

The factors presented in Sections 3.1.1 through 3.1.6 indicate that, at least in the area of knowledge acquisition, which is similar to requirements analysis, new approaches are necessary. The traditional model also imposes a degree of formality that appears inappropriate to an expert system development effort. This is particularly true in the prototyping area.

SECTION 4 - REQUIREMENTS OF AN EXPERT SYSTEM DEVELOPMENT METHODOLOGY

After conducting the research activities described in Section 2 and the analysis described in Section 3, task members arrived at a number of requirements that any expert system methodology must satisfy. These requirements do not constitute a distinct methodology; rather, they represent key elements that such a methodology must contain. Several different methodologies could be derived from these requirements. The list of requirements presented here can be considered a bridge between the research effort discussed in this document and an outline for an expert system development methodology.

Two major assumptions were made. First, the waterfall method is a viable first step for development of an expert system methodology. Second, expert system development comprises a mix of expert system specific activities and traditional software development activities.

As shown in Section 3, there are numerous similarities between activities carried out in the development of an expert system and those of a traditional software project. Because the waterfall method has proven reasonably successful (with the minor problems discussed in Section 3.1.6), there is no reason to abandon the traditional concepts for an expert system methodology. The basic concepts of the waterfall method should be modified as needed to accommodate expert system development activities, such as prototyping.

An expert system consists of software specific to the knowledge base and software that interfaces with the user and other software elements. An expert system can be a part of a larger software system, especially in a typical space

application, or a standalone system. The interfaces are usually written in standard procedural or fourth-generation language, whereas the knowledge base is often written using an expert system shell or other expert system tool.

The requirements that follow are not given in any specific order of importance. The emphasis is on the concepts rather than on the phasing of an activity. An attempt to order this list within the framework of management, technical, and other issues did not prove to be practical; the requirements too often crossed such artificial boundaries.

An expert system development methodology must do the following:

1. Include a special section on the definition of terms. There is a lack of agreement on meanings of commonly used terms or expressions within software engineering. This is particularly true of expert systems. For example, some articles distinguish between a life cycle methodology and a prototyping methodology; others view prototyping as a part of the life cycle. In addition, the relationships among terms must be clear.

2. Establish guidelines that allow evaluation of a problem for its applicability to an expert system solution. During the feasibility study of a new software development effort, certain parts of the entire system (or the whole system itself) should be analyzed for possible implementation as an expert system(s).

3. Include the expert as an active member of the development team, with responsibilities and commitment clearly stated at the beginning of the effort. Initially, the expert provides knowledge to the knowledge engineer through a series of interviews. As the development effort progresses, the role of the expert shifts from a knowledge source to a knowledge base evaluator. As each prototype is

demonstrated, the expert must evaluate the correctness of the knowledge in the knowledge base and the correctness of the solutions provided by the expert system. As a team member, the expert must have duties and responsibilities defined by management (like all other team members) and must perform them willingly.

4. Define the role of each member of the development team. Each member needs to understand his or her duties and responsibilities exactly. Furthermore, each team member needs to be aware of the lines of communication both within and external to the project. Each team member also needs to be aware of the tools available for completing a job.

5. Involve the users with the evaluation of the expert system. The users can range from scientists to operations and maintenance personnel. Each must evaluate the prototype in terms of his or her particular use of the expert system. For example, the scientist would be concerned with ease of use and correctness of a solution; operations personnel would be interested in the ease of maintenance, as well as the ease of operation. As with the expert, the users must be prepared to evaluate the system carefully and objectively. This entails committing time and resources to understand the demonstration materials and the results of the demonstrations and to document comments and concerns. This evaluation will not only result in a system more satisfactory to the user, but it will also lessen necessary user training time upon delivery.

6. Require the assignment of a knowledge engineer with well-defined responsibilities and duties. The goal of the knowledge engineer is to build a system that satisfies both the expert and the user. Thus, the knowledge engineer must acquire as much knowledge as possible about the problem at hand, recognizing again that this requires much time and

effort. Numerous interviews with the expert are necessary, both to obtain information and to ensure that the knowledge engineer understands what is being said. As in the case of the development itself, knowledge acquisition is an iterative process that occurs in parallel with the development of the expert system software. The knowledge engineer must demonstrate the prototype, with special emphasis on closing action items from any previous demonstration. He or she must consider a negative result a useful knowledge acquisition rather than a personal failure; even negative results clarify the requirements.

7. Allow the definition of requirements and development of the expert system software in parallel. This is accomplished through prototyping techniques. A typical iteration of one prototype may include requirements definition/refinement (knowledge acquisition), design, coding, demonstration, and evaluation from experts and users. Each prototype brings the system closer to the completion of requirements specifications and proof of concept. This includes the realization that a selected approach may need to be abandoned and the development effort begun again. In practice, the work will not have been wasted because there is a better understanding of requirements, the relationship between the expert and knowledge engineer has been established, and some aspects of the system may be salvageable.

8. Provide for frequent prototype demonstrations. This fosters a close working relationship between the development team (including the expert) and the users, uniting them in a common goal. The expert and user will have the opportunity to evaluate the correctness of requirements with regard to interpretation and implementation. The knowledge engineers will have the

advantage of frequent feedback from the expert and the users; therefore, they will not waste extensive development time through misinterpretation and will be afforded the opportunity to redesign and reimplement if a better approach is discovered. Managers are assured that progress is being made because the expert, the knowledge engineer, and the users are converging on the definition of requirements and possible knowledge base design. The result of the prototyping effort will be a better, usable system accepted both by the expert and users.

9. Provide for control that allows rapid changes but restricts free-lancing. A way of tracing a system's evolution must be established. The entries to a traceability log must reflect all modifications resulting from new knowledge acquisition (addition, deletion, or changes in requirements), improved understanding of requirements, or a better implementation. Updating such a traceability log must be mandatory but straightforward and fairly simple. Furthermore, this log documents decisions made regarding the expert system, including justifications for the decisions. The importance level of the decisions must be established at the beginning of the project.

Another control log should be created at the initial prototype demonstration and must be reviewed and maintained during the subsequent prototype demonstrations. This log records approval and/or changes to the demonstrated prototype.

10. Contain a set of recommendations on how to schedule expert system development and determine when to finalize the requirements and terminate the prototyping process. This scheduling is expected to become increasingly accurate as data from traceability logs and other technical

documentation from previously developed systems become available for study and analysis. Management on both sides of the project--customer and developer--must agree to an initial estimate of this date at the beginning of the project.

Prototype demonstrations allow frequent reevaluation and calibration of this date according to the actual development process, giving all parties involved in the project a better feel for whether the original cut-off date is feasible or should be replanned.

11. Provide an objective means of reporting progress. During expert system development, the requirements are defined in parallel with the system design and coding, and progress cannot be measured in sequential milestones. Although such terms as progress, productivity, rate of progress, and completion of a schedule milestone still exist, they may have to be redefined to allow for the iterative nature of expert system development. For example, after a prototype demonstration, considerable progress may be made despite a decision to abandon a current expert system development tool and use a new one.

12. Establish a set of baseline documents. The document names, purpose, scope, and intended readers can be taken from the traditional software development methodology (e.g., functional requirements, requirement specifications, operational concepts, user manual, etc.). The first draft of these documents outlining the high-level requirements can be created at the early phases of expert system development. However, they cannot and need not be formally revised and redelivered to keep up with the rate of prototype demonstrations. On the other hand, the updating of the documentation cannot wait until the end of the expert

system development. The solution may lie in the middle: produce revised drafts of the documents reflecting a current "snap shot" state of the requirements definitions and the system design, to be delivered every N demonstrations (or every N months), with a final version delivered at the end of the system development.

In addition to the traditional set of documents, it may be useful to deliver final versions of the system's traceability and control logs for archival purposes.

13. Provide guidelines for scheduling and budgeting that allow for contingency planning (e.g., when a current prototype approach is abandoned).

14. Establish a set of standards and procedures encompassing knowledge acquisition and programming techniques during the actual creation of an expert system, stressing modularity and a structured approach. These standards and procedures must be independent of expert system tools and shells.

15. Allocate an active role for quality assurance during expert system development. Quality assurance functions may differ from those in traditional software development. The quality assurance officer will still have to develop standards and procedures for expert system development; however, procedures to ensure that they are being followed (e.g., inspections) will differ. Frequent prototyping will require an additional balance between routine checking and minimum bureaucracy to ensure that the expert system development process is not stalled. A new function (which may require the involvement of a system engineer) is ensuring that the development goes in the right direction (i.e., the next prototype solves action items from the previous prototype). Thus, quality assurance must participate in every prototype demonstration. A derivative

of this function is the ability to detect lack of progress and to take the appropriate corrective actions. The methodology must provide some guidelines to allow quality assurance to judge whether or not requirements and system development are converging.

16. Establish open communication throughout the project. Management needs to be well informed of progress with the prototypes. The users need to provide feedback on each prototype in a timely fashion. Developers need to talk frequently with each other and with the knowledge engineer. Developers can demonstrate the prototype informally to their peers or to management outside the regularly scheduled prototype demonstration. The expert needs to meet frequently with the knowledge engineer.

SECTION 5 - SUMMARY AND FUTURE GOALS

In assessing the results of the expert system methodology study, the following conclusions were made:

1. Expert system development in industry requires a comprehensive methodology that covers management as well as technical aspects at every phase of an expert system's life cycle. This is also true for expert systems for space applications.

2. No formal methodology currently exists that can be easily, in a straightforward manner, applied in its entirety to expert system development. The waterfall method is used for traditional software development and is widely accepted by managerial and technical participants in the development effort. Being document driven, however, this method is not highly flexible, and is therefore unsuitable for expert system development without modification.

3. Expert systems, although a special class of software with an expert domain that is very well defined but for which requirements cannot be fully expressed algorithmically, are nevertheless software programs. Therefore some aspects of the waterfall method can be applied to the development of an expert system. The prototyping nature of expert system development, combined within the necessity to work closely with experts, requires new methods and approaches.

4. Future work on defining a methodology for expert systems will involve careful analysis of the waterfall method to identify and extract those approaches that are applicable to expert system development requirements. The advantage of this approach is avoiding reinventing the wheel by making full use of proven approaches and methods.

The risk is that the expert system development needs may be inappropriately forced into existing standards and procedures.

5. New standards and procedures will be needed, specifically to address issues of the expert system life cycle (e.g., knowledge acquisition), and the requirements of interfacing traditional and expert system software.

6. Management needs to modify its approach to the control of a software development project. The prototyping technique must not be burdened with inappropriate controls that hamper the development effort. On the other hand, the developers must recognize that prototyping is not a license to disregard proven, useful development practices and controls.

7. Documentation is important throughout the development of the expert system. It must be kept current and must contain reasons why certain actions or designs were implemented rather than others. Early in the development cycle, document structure may be informal to facilitate currency; later, it must be formalized and documents must be placed under strict configuration control.

The next step in the expert system development methodology is to derive a methodology that satisfies the requirements stated in the previous section. It is recommended that the traditional software development approach be used as the starting point and modified where necessary to accommodate the unique expert system development environment. After the expert system development methodology is reasonably well established, suitable projects need to apply all or parts of the methodology. The more controversial aspects of the methodology may be implemented separately for refinement before the entire methodology is applied to a single project.

APPENDIX A - BIBLIOGRAPHY WITH SYNOPSES

This appendix contains the bibliography and synopses of the literature that have been reviewed during the expert system methodology study.

Author: Not specified

Title: Untitled

Source: Presentation by personnel from Inference Incorporated

Synopsis: A presentation on the role of knowledge engineering, the expert system development process, and some heuristics for knowledge engineers.

Author: Not specified

Title: Interview: Peter Hart Talks about Expert Systems

Date: 1986

Journal: IEEE Expert, Vol. 1, No. 1, pp. 96-99

Synopsis: Mr. Hart suggests that an expert system should first be developed as a prototype in LISP in order to allow flexibility in the development of the system and to increase the productivity. After the prototype has been completed, the system can be rewritten in whatever language the developer chooses.

Author: Not specified

Title: MO&DSD Systems Management Policy

Date: 1986

Source: NASA/GSFC, Document MDOD-8YMP/0485

Synopsis: This document defines the systems management policy applicable to all systems developed within the Mission Operations and Data Systems Directorate (MO&DSD) at the Goddard Space Flight Center.

Author: Not specified

Title: MO&DSD Software Development Policy

Date: 1986

Source: NASA/GSFC, Document MODSD-8YDP/0186

Synopsis: This document defines the software development policy applicable to all systems developed within the Mission Operations and Data Systems Directorate at the Goddard Space Flight Center.

Author: Agresti, W. W.

Title: The Conventional Software Life-cycle Model: Its Evolution and Assumptions

Date: 1986

Source: New Paradigms for Software Development, Washington, DC: IEEE Computer Society Press

Synopsis: This tutorial includes 21 articles that discuss the conventional software life-cycle model, its limitations, and possible ways to improve it. The latter include prototyping, operational specification, and transformational implementation.

Authors: Allen, B. P. and Holtzman, P. L.

Title: Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project

Date: August 1987

Source: Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX

Synopsis: The authors describe the Automated Software Development Workstation. The goal of the project is to apply the concept of domain-specific automatic programming systems to

applications at the Lyndon B. Johnson Space Center. Bottlenecks observed in phase I of the project are described. The authors propose to solve these through increased automation of the acquisition of programming knowledge and use of an object-oriented development methodology at all stages of design.

Authors: Bailey, P. A. and Doehr, B. B.

Title: Knowledge Acquisition and Rapid Prototyping of an Expert

System: Dealing with "Real World" Problems

Date: November 13-14, 1986

Source: Conference on Artificial Intelligence for Space Applications, Huntsville, AL

Synopsis: The authors detail their experience with the development of an expert system. In particular, they provide an extensive discussion of the lessons learned during the development and suggestions for expert system development.

Author: Balzer, R.

Title: A 15 Year Perspective on Automatic Programming

Date: 1985

Journal: IEEE Transactions on Software Engineering, Vol. SE-11, No. 11, pp. 1257-1268

Synopsis: The author discusses the standard waterfall method of software development, points out two fundamental flaws that make maintenance difficult, and reviews his efforts taken to alleviate the maintenance problem via automatic programming. The bulk of the effort is in the development of a specification language. Software maintenance is performed by modifying the specification and reimplementing automatically.

Authors: Balzer, R., Cheatham, T. E., and Green, C.

Title: Software Technology in the 1990's: Using a New Paradigm

Date: 1983

Journal: IEEE Computer, Vol. 16, No. 11, 39-45

Synopsis: The authors discuss automatic programming and its advantages during the software life cycle. Less errors will occur since the specification is translated directly into code and testing will become considerably easier.

Author: Bobrow, D. G.

Title: If Prolog Is The Answer, What is the Question or What it Takes to Support the AI Programming Paradigms

Date: 1985

Journal: IEEE Transactions on Software Engineering, Vol. SE-11, No. 11, p. 1401

Synopsis: The author suggests that a single paradigm is not the solution to knowledge programming. Rather, any programming paradigm should be integrated with others into a flexible, user-friendly computing environment.

Authors: Bobrow, D. G., Mittal, S., and Stefik, M. J.

Title: Expert Systems: Perils and Promise

Date: 1986

Journal: Communications of the ACM, Vol. 29, No. 9, pp. 880-894

Synopsis: The paper reviews three successful expert systems (XCON, Pride, and Dipmeter Advisor), provides guidelines for choosing problems suitable for an expert system, and discusses the stages of development for an expert system.

Authors: Bochsler, D. C. and Goodwin, M. A.

Title: A Software Engineering Approach to Expert System Design and Verification

Date: November 13-14, 1986

Source: Conference on Artificial Intelligence for Space Applications, Huntsville, AL, p. 47

Synopsis: This paper presents the methodology used to develop RENEX at the Johnson Space Center. The methodology is molded into that of the standard structured approach. Verification methods are detailed by outlining tests at each step of the verification process.

Author: Boehm, Barry W.

Title: Software Engineering Economics

Date: 1981

Publisher: Englewood Cliffs, NJ: Prentice-Hall, Inc.

Synopsis: This text represents the standard reference for software engineering economics. It is divided into three major parts: an overview of the software life cycle, the fundamentals of software engineering economics, and the art of software cost estimation. The book includes a detailed discussion of the constructive cost model (COCOMO), a model frequently used for estimating and projecting costs for a software development project.

Authors: Bollinger, T, Lightner, E, Laverty, J, Ambrose, E.

Title: The Load Shedding Advisor: An Example of a Crisis-Response Expert System

Date: May 1987

Source: Proceedings of the Goddard Conference on Space Applications of AI and Robotics

Synopsis: The authors discuss a prototype system that can be used to host a load shedding advisor, i. e.,

a system which would monitor major physical environment parameters in a computer facility and recommend appropriate operator responses whenever a serious condition was detected. The expert system is a hybrid that combines procedural, entity-relationship, and rule-based methods. The authors discuss the latter in detail and point out the advantages of this approach.

Author: Boose, J. H.

Title: Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain

Date: December 11-13, 1985

Source: Second Conference on Artificial Intelligence Applications, Miami Beach, pp. 461-466

Synopsis: The author describes a method for combining expertise from many experts in a single domain into a single expert system. Rating grids are used to obtain expertise. These grids are compared. Differences are readily apparent, and the experts can negotiate resolution of differences. The end user, by examining the differences between consensus and dissenting opinions, can readily see the range of acceptable solutions.

Authors: Brauer, D., Roach, P., Frank, M., and Knackstedt, R.

Title: Ada and Knowledge-Based Systems: A Prototype Combining the Best of Both Worlds

Date: October 22-24, 1986

Source: Expert Systems in Government Symposium, McLean, VA, pp. 198-202

Synopsis: The authors discuss the development of the Knowledge-Based Maintenance Expert System (KNOMES). They state that by using Ada as the fundamental structure, they were able to obtain a well-structured, maintainable program and by retaining an artificial intelligence/knowledge-

based language component, they were able to capture the knowledge needed to solve ill-structured, dynamic, and/or nonalgorithmic problems.

Author: Brooks Jr., F. P.

Title: No Silver Bullets: Essence and Accidents of Software Engineering

Date: 1987

Journal: Computer, Vol. 20, No. 4, pp. 10-19

Synopsis: The author looks at the future for software engineering, seeking new breakthroughs for an order of magnitude improvement in productivity, reliability, and simplicity. Among technologies investigated are Ada, object-oriented programming, artificial intelligence, expert systems, automatic programming, graphical programming, programming verification, environments and tools, and workstations. Two of the more promising approaches are incremental development and the use of rapid prototyping for requirements refinement.

Author: Brown, J. S.

Title: The Low Road, the Middle Road, and the High Road

Date: 1984

Source: The AI Business, eds. P. H. Winston, and K. A. Prendergast, pp. 81-90

Publisher: London, England: MIT Press

Synopsis: The author recommends that AI research "travel the low, medium, and high roads simultaneously." The low road puts the intelligence into the programming environment. The middle road encodes experiential knowledge that an expert has accumulated over a period of time or the middle road combines intelligent interfaces with a powerful domain-specific tool. The high road codifies the deep conceptual models.

Author: Brule, J. F.

Title: Expert Systems: Applications

Date: 1986

Source: Artificial Intelligence: Theory, Logic and Application

Publisher: Blue Ridge Summit, PA: Tab Books, Inc.

Synopsis: This chapter discusses the advantages and disadvantages of expert system shells, how to develop an expert system (in general terms), sources of information, and development stages.

Authors: Buchanan, B., Barstow, D., Bechtel, R., Bennett, J., Coancey, W., Kulikowski, C., Mitchell, T., and Waterman, D. A.

Title: Constructing an Expert System

Date: 1983

Source: Building Expert Systems, eds. F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, pp. 127-167

Publisher: Reading, MA: Addison-Wesley

Synopsis: An overview on how to construct an expert system. The authors discuss the evolutionary character of an expert system, identify the major steps in knowledge acquisition, and provide guidelines for constructing an expert system.

Author: Castore, G.

Title: A Formal Approach to Validation and Verification of Knowledge-Based Control Systems

Date: August 1987

Source: Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX

Synopsis: The author describes an approach to the verification and validation of knowledge-based

control systems. His approach is to formulate a structural model for the knowledge-based control system. This model represents a nondeterministic automaton. A logic is then developed for asserting and reasoning about the properties of the structures. Specifications are interpreted as assertions about the model properties. The role of the validation software is to prove these assertions.

Authors: Citrenbaum, R. L. and Geissman, J. R.

Title: A Practical Cost-Conscious Expert System Development Methodology

Date: April 29 - May 1, 1986

Source: AI-86: Artificial Intelligence and Advance Computer Technology Conference, Long Beach, CA

Synopsis: The authors present a three step methodology for developing an expert system. The steps are problem selection, rapid prototype, and iterative growth. They also discuss the staffing profiles and the procedures to be followed by the expert system development team.

Author: Connell, J.

Title: Software Rapid Prototyping

Date: August 27-28, 1987

Source: Seminar presented in Washington, DC

Synopsis: The author presents a structured methodology for rapid prototyping. He works within the framework of the structured techniques of Yourdon and DeMarco. The method requires more time in the traditional requirements analysis phase but gains time in the testing and integration phases since a large part of that effort had been accomplished earlier. According to the author, over the entire life cycle, the new method lowers costs.

Authors: Culbert, C., Riley, G., and Savely, R. T.

Title: Approaches to the Verification of Rule-Based Expert Systems

Date: August 1987

Source: Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX

Synopsis: The authors describe the verification of expert systems. They describe how expert systems are developed, two alternate ways they should be developed, and differences in the verification and validation of conventional software and expert systems.

Authors: Culbert, C., Riley, G., and Savely, R. T.

Title: An Expert System Development Methodology which Supports Verification and Validation

Date: 1987

Source: Paper submitted to 4th IEEE Conference on Artificial Intelligence

Synopsis: The authors present a methodology for the development of an expert system. The phases are problem definition, initial prototype, expanded prototype, delivery/maintenance. The first phase ends with a review by a panel consisting of expert system developers, domain experts, system users, and managers with system responsibility. The panel can become the configuration control board in the fourth phase at the conclusion of testing.

Author: Das, B. K. and Berg, R. A.

Title: Impact Assessment: A Case Study for Scheduling NASA's TDRSS Resources

Date: October 19-23, 1987

Source: Proceedings of Third Annual Expert Systems in Government Conference, Washington, DC, pp. 254-259

Synopsis: The paper outlines the addition of impact assessment to an expert system that can be used for scheduling spacecraft contacts with the Tracking and Data Relay Satellite System (TDRSS). The paper presents the problems of impact assessment and a method that combines model-based reasoning and heuristics used to solve the problems.

Authors: Das, B. K. and Berg, R. A.

Title: Impact Assessment: Perspective of an Operator Assistant for Scheduling NASA's Tracking and Data Relay Satellite System

Date: February 1987

Source: Third IEEE Conference on Artificial Intelligence Applications, Poster Session, Orlando, FL, pp. 23-27

Synopsis: This paper describes an existing prototype system that allows operator's to schedule the TDRSS. The system allows the operator to reschedule activities in an existing schedule. The goal of such a rescheduling is to create minimum impact on the current schedule. The Network Operator Assistant (NOA) accomplishes this goal.

Authors: Das, B. K. and Berg, R. A.

Title: Expert System: A Network Operator Assistant (NOA)

Date: 1987

Source: Unpublished presentation

Synopsis: A slide presentation of NOA (cf. B. K. Das and R. A. Berg, Impact Assessment: Perspective of an Operator Assistant for Scheduling NASA's TDRSS Resources). This presentation provides an overview of NOA and its current status.

Author: DeMarco, T.

Source: Structured Analysis and System Specification

Date: 1979

Publisher: New York: Yourdon, Inc.

Synopsis: This book is the standard reference for conducting structured analysis and system specification. It describes how to draw data flow diagrams, write data dictionary entries and minispecifications. It includes case studies on the technique.

Author: Denning, P. J.

Title: Towards a Science of Expert Systems

Date: 1986

Journal: IEEE Expert, Vol. 1, No. 2, pp. 80-83

Synopsis: A short overview of expert systems. The author discusses the main components of an expert system, deep versus shallow systems, knowledge representation methods, and the limitations of expert systems. He concludes that "there is no reason in principle that, with a different methodology, expert systems cannot be as reliable as other software."

Authors: Dias, W. C., Henricks, J. A., and Wong, J. C.

Title: PLAN-IT: Scheduling Assistant for Solar System Exploration

Date: May 1987

Source: Proceedings of the Goddard Conference on Space Applications of AI and Robotics

Synopsis: The authors discuss the use of PLAN-IT for scheduling activities for the CRAF mission. PLAN-IT is a frame-based system written in Zetalisp. They developed their system in an evolutionary manner, i.e., they started with a basic scheduling system, progressed to a structured system and then, finally, to an expert scheduler.

Author: Dutilly, R.

Title: Automation of Spacecraft Control Centers

Date: May 1987

Source: Proceedings of the Goddard Conference on Space Applications of AI and Robotics

Synopsis: The author discusses the current status of Communications Link Expert Assistance Resource (CLEAR). CLEAR is to be used in the control center at the Goddard Space Flight Center. The author also describes how a series of interacting expert systems could be developed to almost totally automate the Mission Operations Room of the control center. The author provides guidelines and desiderata for such an interactive system.

Author: Ebrahimi, M.

Title: A Structured Approach to Expert System Design

Date: June 2-4, 1987

Source: Proceedings of the Western Conference on Expert Systems, Anaheim, CA, pp. 18-24

Synopsis: The author presents, in general terms, the structured methodology used by Abacus Programming Corporation to develop a design for expert systems.

Authors: Fikes, R. and Kehler, T.

Title: The Role of Frame-Based Representation in Reasoning

Date: 1985

Journal: Communications of the ACM, Vol. 28, No. 9, pp. 904-920

Synopsis: The authors describe the features of frame-based knowledge representation facilities and indicates how they can provide a foundation for a variety of knowledge system functions. They discuss how frames can contribute to knowledge

system reasoning activities and how frames can be used to organize and direct reasoning activities. They also discuss the advantages of integrating frames and production rules into a unified representation facility. One advantage of such a facility is that it makes the organizational and expressive power of object-oriented programming available to domain experts who are not programmers.

Author: Fox, J. M.

Title: AI's Contribution to Software Development or Expert Systems Are Symptoms

Date: October 22-24, 1986

Source: Expert Systems in Government Symposium, McLean, VA, pp. 424-425

Synopsis: The author discusses the application of expert systems to the various steps of the software development life cycle from requirements analysis through integration and testing. He also discusses the applicability to the software manager, systems engineering, quality assurance and configuration control. Finally, he discusses its usage for knowledge-based techniques, his rational for increased productivity, and software prototyping.

Authors: Frail, R. P. and Freedman, R. S.

Title: OPGEN Revisited: Some Methodological Observations on the Delivery of Expert Systems

Date: October 22-24, 1986

Source: Expert Systems in Government Symposium, McLean, VA, pp. 310-317

Synopsis: After a discussion of OPGEN, the authors provide five observations and guidelines associated with the delivery of expert systems. The observations are:

1. The delivery of an expert system should be considered in the earliest stage of development.

2. Production rules are useful for rapidly prototyping a small system for the purpose of demonstrating feasibility.

3. Adding rules does not necessarily improve an expert system.

4. If an expert system product has been successfully used to solve problems in several separate, but similar domains, then it may be cost-effective to transform that system into a shell.

5. Is the final deliverable system still an expert system?

Author: Friedland, P.

Title: Architectures for Knowledge-Based Systems

Date: 1985

Journal: Communications of the ACM, Vol. 28, No. 9, p. 903

Synopsis: This is a one page introduction to three articles that describe the different ways to represent knowledge in an expert system: frames, rules, and logic (cf. R. Fikes and T. Kehler, 1985, F. Hayes-Roth, 1985, and M. R. Genesereth and M. L. Ginsberg, 1985). Friedland emphasizes that these are not three competing methods, but rather choices available for different applications.

Authors: Gaschnig, J., Klahr, P., Pople, H., Shortliffe, E., and Terry, A.

Title: Evaluation of Expert Systems: Issues and Case Studies

Date: 1983

Source: Building Expert Systems, eds. F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, pp. 241-280

Publisher: Reading, MA: Addison-Wesley

Synopsis: The authors discuss the need for testing expert systems, and when and what to test. They provide a checklist of pitfalls that must be avoided in the preparation of testing. They also provide two case studies: R1 (XCON) and a chemical spill problem. (The latter is a hypothetical problem used throughout the book.)

Authors: Genesereth, M. R. and Ginsberg, M. L.

Title: Logic Programming

Date: 1985

Journal: Communications of the ACM, Vol. 28, No. 9, pp. 933-941

Synopsis: The authors have written an introduction to logic programming and explain concepts in a straightforward, understandable manner. They describe both advantages and shortcomings of logic programming. The authors are optimistic that logic programming is emerging as the dominant programming method for the next century.

Author: Gilb, T.

Title: Evolutionary Delivery versus the "Waterfall Model"

Date: 1985

Journal: ACM SIGSOFT Software Engineering Notes, Vol. 10, No. 3, pp. 49-61

Synopsis: The author proposes the evolutionary delivery as a replacement for the standard waterfall model of software development. The evolutionary delivery method is based on the following principles: deliver something to a real end-user; measure the added-value to the user; and adjust both design and objectives based on observations. The article basically describes the author's approach to prototyping.

Author: Gladden, G. R.

Title: Stop the Life-Cycle, I Want to Get Off

Date: 1982

Journal: ACM SIGSOFT, SW Engineering Notes, Vol. 7, No. 2, pp. 35-39

Synopsis: The author discusses the problems of the traditional waterfall model and proposes the "Hollywood" approach to software development. This approach, which uses rapid prototyping, is based on the following three propositions: 1) system objectives are more important than system requirements; 2) a physical object conveys more information that a written specification; 3) system objectives plus physical demonstrations will result in a successful product.

Author: Goldberg, A. T.

Title: Knowledge-based Programming: A Survey of Program Design and Construction Techniques

Date: 1986

Journal: IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, pp. 752-768

Synopsis: The author discusses data structure selection techniques, procedural representation of logic assertions, store-versus-compute tradeoffs, finite differencing, loop fusion, and algorithm design methods.

Authors: Goodwin, M. A. and Robertson, C. C.

Title: Expert System Verification Concerns in an Operations Environment

Date: August 1987

Source: Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX

Synopsis: The authors provide the definition of a verified expert system, discuss lessons learned from the past for the verification of conventional software, and extend these techniques to operational expert systems.

Authors: Green, C. J. R. and Keyes, M. M.

Title: Verification and Validation of Expert Systems

Date: June 2-4, 1987

Source: Proceedings of the Western Conference on Expert Systems, Anaheim, CA, pp. 38-43

Synopsis: The authors propose a method for the verification and validation of expert systems. As in conventional software development, the verification and validation of expert systems will comprise requirements definition, verification, test case preparation, test execution, and evaluation. Differences between the proposed method and conventional verification and validation are distinguished.

Author: Hancock, B.

Title: Expert Systems

Date: 1987

Journal: DEC Professional, Vol. 6, No. 5, pp. 40-48

Synopsis: An overview of expert systems that describes five approaches to knowledge representation, a five-phased approach to expert system development, and applications of expert systems.

Authors: Hartrum, T. C. and Lamont, G. B.

Title: Development of a Comprehensive Software Engineering Environment

Date: August 1987

Source: Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX

Synopsis: The authors describe a software engineering environment under development, and partially in use, at the Air Force Institute of Technology. The goal is to integrate tools and techniques across all phases of the DoD 2167 life cycle. The central concept to their environment is that the data dictionary provides the complete description of the entire development effort.

Author: Hayes, C.

Title: Using Goal Interactions to Guide Planning

Date: July 13-17, 1987

Source: Proceedings of the Sixth National Conference on AI, Seattle, WA, Vol. 1, pp. 224-228

Synopsis: The author describes the program Machinist. This program is an improvement over conventional planning programs. Before generating any plans, it looks for cues in the problem specification that may indicate potential difficulties or conflicting goal interactions. Machinist plans around these difficulties, greatly increasing the probability of producing a good plan on the first try.

Author: Hayes-Roth, F.

Title: Rule-Based Systems

Date: 1985

Journal: Communications of the ACM, Vol. 28, No. 9, pp. 921-932

Synopsis: The author presents an overview of rule-based systems including the architecture, technology evolution, and the strengths and weaknesses for such systems.

Author: Hayes-Roth, F.

Title: The Knowledge-Based Expert System: A Tutorial

Date: 1984

Journal: Computer, Vol. 17, No. 9, pp. 11-28

Synopsis: The author provides an overview of knowledge engineering, a discussion of the principal scientific and engineering issues in the field, the process of building an expert system, the role of tools in that process, how expert systems perform human-computer interface functions, and the frontiers of research and development.

Author: Hull, L. G.

Title: Managing an Expert System Project

Date: 1986

Source: A NASA/GSFC presentation

Synopsis: Slides from a GSFC presentation. Discusses expert system technology, methodology, management, and testing. Compares expert system and standard software development methodologies.

Author: Jackson, R.

Title: Expert Systems Built by the "Expert": An Evaluation of OPS5

Date: May 1987

Source: Proceedings of the Goddard Conference on Space Applications of AI and Robotics

Synopsis: The author relates his experiences with the development of two expert systems using OPS5. The author feels that OPS5 was easy to learn and allowed easy modification of his two small systems.

Authors: Jacob, R. J. K. and Froscher, J. N.

Title: Developing a Software Engineering Methodology for Knowledge-Based Systems

Date: 1986

Source: Naval Research Laboratory Report 9019

Synopsis: The authors describe a design methodology intended to make a knowledge-based system easier to change. The method divides the domain knowledge into groups and attempts to limit carefully and specify formally the flow of information between these groups to localize the effects of typical changes within the groups.

Authors: Jenkins, J. P. and Nelson, W. R.

Title: Expert Systems and Accident Management

Date: October 22-24, 1986

Source: Expert Systems in Government Symposium, McLean, VA, pp. 88-94

Synopsis: The authors discuss the Accident Management Expert System (AMES) which is developed from a deep knowledge base and characterized by success paths in the form of a response tree. They also describe an experiment in which trained operators handled a simulated emergency with and without an expert system as an aid; an autonomous expert system also handled the same emergency. The experiment revealed that the unaided operator and the autonomous expert system performed equally well while the operator using an expert system as an aid took longer to handle the emergency and did not perform aswell. The authors conclude by describing how to develop a deep knowledge base and an expert system for the nuclear power plant industry.

Authors: Kass, R. and Finin, T.

Title: Rules for the Implicit Acquisition of Knowledge about the User

Date: July 13-14, 1987

Source: Proceedings for the Sixth National Conference on AI, Seattle, WA, Vol. 1, pp. 295-300

Synopsis: The authors discuss techniques for acquiring knowledge about the user implicit in interactions between users and cooperative advisory systems. They developed these techniques by analyzing transcripts of a large number of interactions between advice-seekers and a human expert, and have encoded them as a set of user model acquisition rules.

Author: Keller, R.

Title: Expert System Technology: Development and Application

Date: 1987

Publisher: Englewood Cliffs, NJ: Yourdon Press

Synopsis: The author introduces expert systems to managers and computer professionals. He incorporates these systems into existing environments, i.e., he shows how existing software development methodologies can be modified to encompass expert systems. Considerable emphasis is placed on structured analysis. Detailed examples are provided.

Authors: Kelly, V. E. and Nonnenmann, U.

Title: Inferring Formal Software Specifications From Episodic Descriptions

Date: July 13-17, 1987

Source: Proceedings of the Sixth National Conference on AI, Seattle, WA, Vol. 1, pp. 127-132

Synopsis: The authors describe WATSON, a system that converts informal, incomplete requirements into formal specifications that are consistent, complete for a given level, and executable.

Authors: Kitto, C. M. and Boose, J. H.

Title: Choosing Knowledge Acquisition Strategies for Application Tasks

Date: June 2-4, 1987

Source: Proceedings of the Western Conference on Expert Systems, Anaheim, CA, pp. 96-103

Synopsis: The authors describe a method for providing automated assistance to the knowledge engineer or domain expert in analyzing the problem domain, classifying the problem tasks and subtasks, identifying problem-solving methods, suggesting knowledge acquisition tools, and recommending specific strategies for knowledge acquisition within those tools.

Authors: Kline, P. J. and Dolins, S. B.

Title: Choosing Architectures for Expert Systems

Date: 1985

Source: US Department of Commerce, NTIS, Springfield, VA

Synopsis: This work documents guidelines that may be used in designing expert systems. The 47 guidelines presented were derived from a search of published literature and were reviewed by experienced expert system builders. In addition, the report contains 44 problems or pitfalls that may be encountered with the implementation techniques.

Authors: Kline, P. J. and Dolins, S. B.

Title: Problem Features That Influence the Design of Expert Systems

Date: 1986

Source: AAAI 86: Fifth National Conference on Artificial Intelligence, pp. 956-962

Synopsis: The authors discuss five general problems that are important for the proper use of a wide variety of artificial intelligent implementation techniques. Knowledge engineers can improve the likelihood of obtaining the right design for an expert system if they are aware of these problem features.

Author: Kowalski, R.

Title: AI and Software Engineering

Date: 1984

Journal: Datamation, Vol. 30, No. 18, pp. 92-102

Synopsis: The author suggests that software development can improve by taking advantage of the new technology available to it. Specifically, logic programming can be worked within the framework of structured analysis and structured design to eliminate the need for the implementation phase of software development.

Authors: Kozick Jr., B. and Reynolds, W.

Title: MPAD MCC Workstation System: ADDAM and EEVE

Date: August 1987

Source: Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX

Synopsis: The authors describe the three phases of the data workstation development for the automated data distribution and management (ADDAM) system and the design of an associated monitoring and control expert system, the Effective Evaluation (EEVE), that is responsible for ensuring data integrity.

Author: Kraft, A.

Title: XCON: An Expert Configuration System at Digital Equipment Corporation

Date: 1984

Source: The AI Business, eds. P. H. Winston and K. A. Prendergast, pp. 41-49

Synopsis: The author provides a history and status report on XCON.

Authors: Lansky, A. L. and Fogelsong, D. S.

Title: Localized Representation and Planning Methods for Parallel Domains

Date: July 13-17, 1987

Source: Proceedings of the Sixth National Conference on AI, Seattle, WA, Vol. 1, pp. 240-245

Synopsis: The authors present a general method for structuring domains that is based on the concept of locality. They consider a localized domain description to be one that is partitioned into regions of activity. They present an overview of the Group Element Model (GEM), and discuss GEMPLAN, a localized planner based on the GEM representation.

Author: Lantz, K.

Title: The Prototyping Methodology: Designing Right the First Time

Date: April 7, 1986

Journal: Computerworld, pp. 69-72

Synopsis: The author offers the prototyping methodology as an approach to software design. He discusses the problems associated with the traditional waterfall method and the advantages of prototyping. He emphasizes that prototyping is not synonymous with hacking but is a viable method for producing a good product in less time than required by the traditional method.

Author: Leinweber, D.

Title: Expert Systems in Space

Date: 1987

Journal: IEEE Expert, Vol. 2, No. 1, pp. 26-36

Synopsis: In this article the author discusses the potential role of expert systems as monitors and real-time controllers of space platforms. He also discusses the use of PICON (process intelligent control), an expert system tool specifically designed to build real-time systems.

Authors: Lindenmayer, K., Vick, S., and Rosenthal, D.

Title: Maintaining an Expert System for the Hubble Space Telescope Ground Support

Date: May 13-14, 1987

Source: Proceedings of the Goddard Conference on Space Applications of AI and Robotics

Synopsis: This paper describes maintenance issues associated with the coupling of rules within a rule-based system, Transformation, and offers a method for partitioning a rule base so that the amount of knowledge needed to modify the rule base is minimized.

Authors: Lines-Browning, K. M. and Stone, J. L.

Title: An Expert System That Performs A Satellite Stationkeeping Maneuver

Date: May 1987

Source: Proceedings of the Goddard Conference on Space Applications of AI and Robotics

Synopsis: The authors discuss the prototype Expert System for Satellite Orbit Control (ESSOC). In addition to the details of ESSOC, they discuss their development methodology (rapid prototyping) and testing philosophy.

Authors: Lubars, M. D. and Harandi, M. T.

Title: Intelligent Support for Software Specification and Design

Date: 1986

Journal: IEEE Expert, Vol. 1, No. 4, pp. 33-41

Synopsis: The authors discuss how structured techniques can be combined with Intelligent Design Aid (IDeA) to develop software. IDEA is an environment for supporting high-level software specification and design. In addition, analysis and implementation are conducted in parallel.

Author: Martin, N.

Title: The Management of Expert System Development

Date: October 1987

Source: Tutorial presented at IEEE Expert Systems in Government Conference, Washington, DC

Synopsis: Nancy Martin provides a description of her Expert System Controlled Iterative Enhancement (ESCIE). She includes in this tutorial the cost and schedule drivers of an expert system, the expert system development team, project planning, and acceptance criteria for an expert system.

Author: Medeiros, E.

Title: Steps to a New Methodology

Date: 1987

Source: Unpublished manuscript

Synopsis: A suggested approach to the development of an expert system within the framework of structured analysis and design.

Author: Myers, W.

Title: Introduction to Expert Systems

Date: 1986

Journal: IEEE Expert, Vol. 1, No. 2, pp. 100-109

Synopsis: The author describes the fundamentals of expert systems, the languages and hardware used in artificial intelligence, and software development tools. He provides a description of software development tools for large machines as well as for personal computers.

Authors: Narayanaswamy, K. and Scacchi, W.

Title: Maintaining Configurations of Evolving Software Systems

Date: 1987

Journal: IEEE Transactions on Software Engineering, Vol. SE-13, No. 3, pp. 324-334

Synopsis: The authors discuss an approach to maintaining the configuration of evolving software systems. They propose that software be structured into module families and suggest the use of NuMIL to describe software system configuration and use of a prototype environment they developed to maintain software system configuration.

Authors: Nguyen, T. A., Perkins, W. A., Laffey, T. J., and Pecora, D.

Title: Knowledge Base Verification

Date: 1987

Journal: AI Magazine, Vol. 8, No. 2, pp. 69-75

Synopsis: The authors describe CHECK, a computer program, that implements an algorithm to verify the consistency and completeness of knowledge bases for the Lockheed expert system shell.

Author: O'Bannon, R. M.

Title: An Intelligent Aid to Assist Knowledge Engineers with Interviewing Experts

Date: June 2-4, 1987

Source: Proceedings of the Western Conference on Expert Systems, Anaheim, CA, pp. 31-36

Synopsis: The author describes a program that is an intelligent, interactive aid designed to assist knowledge engineers with eliciting, recording, and analyzing the responses of an expert.

Authors: O'Keefe, R. M., Balci, O., and Smith, E. P.

Title: Validating Expert System Performance

Date: 1987

Journal: IEEE Expert, Vol. 2, No. 4, pp. 81-90

Synopsis: The authors discuss the importance of validation and evaluation in the research and development of expert systems and methods for formal validation of expert systems.

Author: Partridge, D.

Title: Engineering Artificial Intelligence Software

Date: 1986

Journal: Artificial Intelligence Review, Vol. 1, pp. 27-41

Synopsis: The author discusses the need for a methodology for AI program development, the current differences in how AI and conventional software programs are developed, and the restrictions of current methodologies that prevent AI from reaching its full potential. The author does advocate the use of incremental development but warns against the tendency towards hacking. Judicious use of incremental development is required.

Author: Patrick, M.

Title: Surprise Control

Date: November 16, 1987

Journal: Computerworld, Vol. 21, pp. 93-98

Synopsis: The author discusses the importance of including the end user in the testing process and of developing software in an iterative life cycle. He also stresses the importance of developing test plans in parallel with the corresponding development products.

Author: Prerau, D. S.

Title: Knowledge Acquisition in the Development of a Large Expert System

Date: 1987

Journal: AI Magazine, Vol. 8, No. 2, pp. 43-51

Synopsis: The author discusses over 30 points on knowledge acquisition that were found to be important during the development of the Central Office Maintenance Printout Analysis and Suggestion System (COMPASS). The guidelines cover selecting an expert and an appropriate domain for the expert system, getting started in knowledge acquisition, documenting the knowledge, and actually acquiring and recording the knowledge.

Author: Ram, A.

Title: AQUA: Asking Questions and Understanding Answers

Date: July 13-17, 1987

Source: Proceedings of the Sixth National Conference on AI, Seattle, WA, Vol. 1, pp. 312-316

Synopsis: The author presents an understanding algorithm based on the theory that readers ask questions as they read. The algorithm is a three-step process: read, explain, and generalize. The author has implemented a computer program, AQUA, that embodies his theory of questions and understanding.

Authors: Ramamoorthy, C., Prakash, A., Tsai, W., Usuda, Y.

Title: Software Engineering: Problems and Perspectives

Date: 1984

Journal: Computer, Vol. 17, No. 10, pp. 191-209

Synopsis: The authors investigate various subjects of software engineering by reviewing their activities, their importance, and new directions in software engineering. The subjects cover all aspects of the traditional life cycle as well as metrics, cost estimation, rapid prototyping, and quality assurance.

Authors: Richardson, K. and Wong, C.

Title: Knowledge Based System Verification and Validation as Related to Automation of Space Station Subsystems: Rationale for a Knowledge Based System Life Cycle

Date: November 2-3, 1987

Source: Proceedings of the Third Conference on Artificial Intelligence for Space Applications, NASA Conference Publication 2492, Huntsville, AL

Synopsis: The authors present a life cycle that is to be used for knowledge-based systems. It is designed to solve some problems of the verification and validation of such systems. The life cycle consists of the following phases: requirements, prototype, knowledge-based system build, test, and delivery and monitor.

Author: Ringland, G.

Title: Software Engineering in a Development Group

Date: 1984

Journal: Software-Practice and Experience, Vol. 14, No. 6, pp. 533-559

Synopsis: The author discusses his experience in managing several small software projects. He provides productivity data for each of the projects and compares them to those predicted by Boehm's algorithm. He discusses productivity relative to the reuse of software, maintenance and enhancement, and documentation.

Authors: Robinson, J. A. and Satterlee, A. A.

Title: An In-Service Expert System for Configuring Thrusters on Orbiting Spacecraft

Date: June 2-4, 1987

Source: Proceedings of the Western Conference on Expert Systems, Anaheim, CA, pp. 112-117

Synopsis: The authors describe a program that is an intelligent, interactive aid designed to assist knowledge engineers with the eliciting, recording, and analyzing the responses of an expert.

Author: Rolston, D. W.

Title: An Expert System for Reducing Software Maintenance Costs

Date: October 22-24, 1986

Source: Expert Systems in Government Symposium, McLean, VA, pp. 396-406

Synopsis: The author describes Problem Resolution System (PRESS), an expert system that is designed to reduce software maintenance costs by analyzing information associated with problem reports to solve software problems. Discussed in detail are the definition of the problem, an overview of the problem solution, the design of PRESS, a description of the development methodology, and the technique used for gathering statistics that are used to evaluate PRESS.

Authors: Rook, F. W. and Odubiyi, J. B.

Title: An Expert System for Satellite Orbit Control (ESSOC)

Date: October 22-24, 1986

Source: Expert Systems in Government Symposium, McLean, VA, pp. 221-227

Synopsis: The authors discuss the design and development of an expert system to aid in the ground-based satellite control process. Each of the following are discussed in detail: the problem domain, design methodology, satellite environment, and the development environment. In addition, the functions performed by ESSOC are described as well as the operation of the expert system. The latter discusses the knowledge representation scheme, the inferencing cycle, and the user interface.

Author: Schneidewind, N. F.

Title: The State of Software Maintenance

Date: 1987

Journal: IEEE Transactions on Software Engineering, Vol. SE-13, No. 3, pp. 303-310

Synopsis: The author reviews the state of software maintenance. Aspects of maintenance discussed are "the maintenance problem", models for maintenance, methods to improve maintenance, maintenance information management, maintenance standards, and maintenance of existing code.

Author: Simon, H. A.

Title: Whether Software Engineering Needs to Be Artificially Intelligent

Date: 1986

Journal: IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, pp. 726-732

Synopsis: The author points out that artificial intelligence and software engineering have to work together to make artificial intelligence a success.

Authors: Soloway, E., Bachant, J., and Jensen, K.

Title: Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY Large Rule-Base

Date: July 13-17, 1987

Source: Proceedings of the Sixth National Conference on AI, Seattle, WA, Vol. 2, pp. 824-829

Synopsis: The article describes the problems encountered in maintaining XCON, an expert system with very large rule base, and proposes solutions to these problems. The authors are using XCON-in-RIME, a higher order language than XCON and a programming tool called SEAR that "provides on-line enforcement of coding guidelines." The guidelines correspond to structured programming practices.

Authors: Stachowitz, R. A. and Combs J. B.

Title: Validation of Expert Systems

Date: January 6-9, 1987

Source: Proceeding of the Hawaii International Conference on Systems Sciences, Kona, Hawaii

Synopsis: A description of the Expert System Validation Associate (EVA) is provided by the authors. This tool is under development at the Lockheed Missiles and Space Company, Inc. The purpose of EVA is to define and develop automated tools to validate the structural, logical, and semantic integrity of expert systems.

Authors: Stachowitz, R. A., Chang, C. L., Stock, T. S., and Combs, J. B.

Title: Building Validation Tools for Knowledge-Based Systems

Source: Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX

Synopsis: The authors describe the Expert Systems Validation Associate (EVA), a knowledge-based system that is designed to improve the validation process by finding mistakes and omissions in the knowledge base, by proposing knowledge base extensions and modifications, and by showing the impact of changes to the knowledge base.

Authors: Steppel, S., Clark, T. L., Belford, P. C., Bumgarner, W. D., Federoff, A. M., Frankle, N. S., Torreele, J. A., and Zuccaro, J. A.

Title: Digital System Development Methodology

Date: 1985

Synopsis: Digital System Development Methodology (DSDM®) defines the practices used by Computer Sciences Corporation to develop systems. It covers all aspects of system development, from management to systems integration and testing.

Author: Subrahmanyam, P. A.

Title: The "Software Engineering" of Expert Systems: Is Prolog Appropriate?

Date: 1985

Journal: IEEE Transactions on Software Engineering, Vol. SE-11, No. 11, pp. 1391-1400

Synopsis: The author examines the basic features of Prolog from different viewpoints to determine whether these features support expert system development. The author concludes that, while useful, Prolog needs to be modified and enhanced.

Authors: Swigger, K., Burns, H., Loveland, H., and Jackson, T.

Title: An Intelligent Tutoring System for Interpreting Ground Tracks

Date: July 13-17, 1987

Source: Proceedings of the Sixth National Conference on AI, Seattle, WA, Vol. 1, pp. 72-76

Synopsis: The authors describe an intelligent tutoring system that helps students learn how ground tracks are related to the orbital elements of space craft. The program trains students to estimate orbital parameters from a ground track. The program allows the student to change parameters to determine the effect of the altered parameter on the ground track.

Author: Teitelman, W.

Title: A Tour Through Cedar

Date: 1985

Journal: IEEE Transactions on Software Vol. SE-11, No. 3, pp. 285-302

Synopsis: The author describes the Cedar programming environment that combines high-quality graphics, a sophisticated editor and document preparation facility, and a variety of tools for the programmer to use in the construction and debugging of programs in a single integrated environment.

Authors: van de Brug, A., Bachant, J., and McDermott, J.

Title: The Taming of R1

Date: 1986

Journal: IEEE Expert, Vol. 1, No. 3, pp. 33-39

Synopsis: The authors discuss the difficulties of adding knowledge to R1 (XCON) and the use of Rime to solve this problem.

Authors: Walker, T. C. Miller, R. K.

Title: Expert Systems 1986

Date: 1986

Publisher: Madison, GA: SEAI Technical Publications

Synopsis: This volume is a gold mine of information about expert systems. It provides information on the technology of expert systems, tools for building expert systems, and summarizes industrial, commercial, professional, and military, aerospace, and transportation applications.

Author: Waterman, D. A.

Title: A Guide to Expert Systems

Date: 1986

Publisher: Reading, MA: Addison-Wesley

Synopsis: The book is divided into six sections: 1. Introduction to Expert Systems, 2. Expert System Tools, 3. Building an Expert System, 4. Difficulties with Expert System Development, 5. Expert Systems in the Marketplace, and 6. Expert Systems and Tools. The book is well written and very understandable. It provides a good introduction to expert systems in that it provides lucid descriptions of expert systems as well as information about current tools and existing systems.

Author: Wingert, W. B.

Title: Verifying Shuttle Onboard Software Using Expert Systems

Date: August 1987

Source: Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX

Synopsis: The author describes the prototype version of the Analysis Criteria Expert System (ACES) which verifies the shuttle onboard flight software. The types of testing are described as well as experiences learned during the implementation of the prototype. Details of the verification are provided.

Authors: Woolf, B. and Cunningham, P.

Title: Building A Community for Intelligent Tutoring Systems

Date: July 13-17, 1987

Source: Proceedings of the Sixth National Conference on AI, Seattle, WA, Vol. 1, pp. 82-87

Synopsis: The authors discuss the need for multiple experts to work together to develop knowledge

representation systems for intelligent tutors. They describe example methodologies for building tools for knowledge acquisition. This includes specific tasks and criteria that might be used to transfer expertise from several experts to an intelligent tutoring system.

Author: Zack, Barry A.

Title: Building Operational Expert Systems

Date: October 1987

Source: Tutorial presented at IEEE Expert Systems in Government Conference, Washington, DC

Synopsis: This tutorial describes the functions involved with building an expert system. Subjects covered are knowledge engineering, development and delivery environments, the building, evaluation, and deployment of knowledge-based systems.

Author: Zualkernan, I., Tsai, W. T., and Volovik, D.

Title: Expert Systems and Software Engineering: Ready for Marriage?

Date: 1986

Journal: IEEE Expert, Vol. 1, No. 4, pp. 24-31

Synopsis: The authors discuss the waterfall model of software development and then discuss how expert systems can be applied to the different phases. They provide a case study in software testing.

APPENDIX B - REFERENCE EVALUATION

This appendix provides the evaluation of the characteristics of each reference read for this study. A weighted code, from 1 to 5 (with 5 representing the highest value), was assigned to each reference based on its relevance to the study. No entry is made in the table for those references that could not be placed easily into any of the classification criteria. The characteristics are planning (PLN), organizing (ORG), monitoring (MON), and controlling (CTR) in the management area and quality assurance (QA), configuration control (CC), scheduling (SCH), and development life cycle (DC) in the technical area.

		MANAGEMENT				TECHNICAL			
ID	TITLE	PLN	ORG	MON	CTR	QA	CC	SCH	DC
1	A Guide to Expert Systems								3
2	Managing an Expert System Project								5
3	Constructing an Expert System								5
4	Expert Systems								4
5	Expert System Technology: Development and Application								4
6	Impact Assessment: Perspective of an Operator Assistant for Scheduling NASA's TDRSS								3
7	Expert System: A Network Operator Assistant (NOA)								2
8	Knowledge-based Programming: A Survey of Program Design & Construction Techniques								1

ID	TITLE	MANAGEMENT PLN	ORG	MON	CTR	TECHNICAL QA	CC	SCH	DC
9	Maintaining an Expert System for the HST Ground Support								4
10	Architectures for Knowledge-Based Systems								3
11	Towards a Science of Expert Systems				5				5
12	XCON: An Expert Configuration System at Digital Equipment Corporation								2
13	The Low Road, the Middle Road, and the High Road								3
14	Evaluation of Expert Systems: Issues and Case Studies								4
15	Whether Software Engineering Needs to Be Artificially Intelligent								3
16	Interview: Peter Hart Talks about Expert Systems								
17	The Knowledge-Based Expert System: A Tutorial								4
18	Introduction to Expert Systems								4
19	Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain								
20	The State of Software Maintenance								
21	Maintaining Configurations of Evolving Software Systems		2						

ID	TITLE	MANAGEMENT PLN	ORG	MON	CTR	TECHNICAL QA	CC	SCH	DC
22	A 15 Year Perspective on Automatic Programming				3				2
23	An Intelligent Tutoring System for Interpreting Ground Tracks								
24	Building A Community for Intelligent Tutoring Systems				5				
25	Inferring Formal Software Specifications From Episodic Descriptions								4
26	Using Goal Interactions to Guide Planning			3					
27	Localized Representation and Planning Methods for Parallel Domains							3	
28	Rules for the Implicit Acquisition of Knowledge about the User								4
29	Expert Systems 1986								
30	AQUA: Asking Questions and Understanding Answers								2
31	The "Software Engineering" of Expert Systems: Is Prolog Appropriate?								3
32	If Prolog Is The Answer, What is the Question? or What it Takes to Support the AI Programming Paradigms								4
33	The Role of Frame-Based Representation in Reasoning								
34	Logic Programming								

ID	TITLE	MANAGEMENT PLN	ORG	MON	CTR	TECHNICAL QA	CC	SCH	DC
35	Expert Systems: Applications								5
36	A Software Engineering Approach to Expert System Design and Verification								5
37	Steps to a New Methodology								3
38	untitled								5
39	Impact Assessment: A Case Study for Scheduling NASA's TDRSS Resources								2
40	Expert Systems: Perils and Promise								4
41	Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a Very Large Rule-Base								5
42	Rule-Based Systems								
43	Software Engineering in a Development Group				3				3
44	Developing a Software Engineering Methodology for Knowledged-Based Systems						4		4
45	MO&DSD Systems Management Policy								5
46	MO&DSD Software Development Policy								5
47	Digital System Development Methodology								5
48	A Tour Through Cedar								1

ID	TITLE	MANAGEMENT PLN	ORG	MON	CTR	TECHNICAL QA	CC	SCH	DC
49	Expert Systems and Software Engineering: Ready for Marriage?								5
50	Intelligent Support for Software Specification and Design								5
51	OPGEN Revisited: Some Methodological Observations on the Delivery of ES								4
52	Expert Systems Built by the "Expert": An Evaluation of OPS5								3
53	Engineering Artificial Intelligence Software								5
54	AI's Contribution to Software Development or Expert Systems are Symptoms								
55	PLAN-IT: Scheduling Assistant for Solar System Exploration								4
56	An Expert System That Performs A Satellite Stationkeeping Maneuver						5		5
57	The Load Shedding Advisor: An Example of a Crisis-Response Expert System								4
58	Automation of Spacecraft Control Centers								
59	Expert Systems and Accident Management								4
60	Ada and Knowledge-Base Systems: A Prototype Combining the Best of Both Worlds								3

ID	TITLE	MANAGEMENT PLN	ORG	MON	CTR	TECHNICAL QA	CC	SCH	DC
61	An Expert System for Satellite Orbit Control (ESSOC)								5
62	An Expert System for Reducing Software Maintenance Costs								
63	An In-Service Expert System for Configuring Thrusters on Orbiting Spacecraft								
64	An Intelligent Aid to Assist Knowledge Engineers with Interviewing Experts								
65	Choosing Knowledge Acquisition Strategies for Application Tasks								
66	Verification and Validation of Expert Systems								5
67	A Structured Approach to Expert System Design								5
68	The Prototyping Methodology: Designing Right the First Time								5
69	Stop the Life-Cycle, I Want to Get Off								5
70	Software Technology in the 1990's: Using a New Paradigm								2
71	Evolutionary Delivery versus the "Waterfall Model"								5
72	Expert Systems in Space								
73	The Taming of R1						3		

ID	TITLE	MANAGEMENT PLN	ORG	MON	CTR	TECHNICAL QA	CC	SCH	DC
74	AI and Software Engineering								4
75	Software Engineering: Problems and Perspectives								3
76	Knowledge Acquisition and Rapid Prototyping of an Expert System: Dealing with "Real World" Problems								4
77	The Conventional Software Life-cycle Model: Its Evolution and Assumptions								4
78	Defense Science Board Final Report on the Software Task Force								3
79	Defense System Software Development					3	3	3	3
80	Rapid Prototyping	3	3						5
81	Structured Analysis and System Specification								4
82	Choosing Architectures for Expert Systems								4
83	Software Engineering Economics	4	4	4	4	4	4	4	4
84	Building Operational Expert Systems	4	4						4
85	The Management of Expert System Development	5	5	4	5			5	5
86	No Silver Bullets: Essence and Accidents of Software Engineering								3

ID	TITLE	MANAGEMENT PLN	ORG	MON	CTR	TECHNICAL QA	CC	SCH	DC
87	Development of a Comprehensive Software Engineering Environment								4
88	Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project								3
89	Verifying Shuttle Onboard Software Using Expert Systems						5		5
90	Building Validation Tools for Knowledge-Based Systems								3
91	Expert System Verification Concerns in an Operations Environment								5
92	MPAD MCC Workstation System: ADDAM and EEVE								
93	Approaches to the Verification of Rule-Based Expert Systems								5
94	A Formal Approach to Validation and Verification of Knowledge-Based Control Systems								4
95	An Expert System Development Methodology which Supports Verification and Validation								5
96	A Practical Cost-Conscious Expert System Development Methodology								4

ID	TITLE	MANAGEMENT PLN	ORG	MON	CTR	TECHNICAL QA	CC	SCH	DC
97	Validation of Expert Systems								2
98	Validating Expert System Performance								5
99	Surprise Control								3
100	Problem Features That Influence the Design of Expert Systems								4
101	Knowledge Acquisition in the Development of a Large Expert System								5
102	Knowledge Base Verification								2
103	Knowledge Based System Verification and Validation as Related to Automation of Space Station Subsystems: Rationale for a Knowledge Based System Life cycle								5

APPENDIX C - LIST OF INTERVIEWEES

L. Botten - Computer Sciences Corporation
Systems Sciences Division
8728 Colesville Road
Silver Spring, MD 20910

J. Buser - 508B Country Club Parkway
Mt. Laurel, NJ 08054

J. Bush - Computer Sciences Corporation
Systems Sciences Division
4600 Powder Mill Road
Beltsville, MD 20705

A. Critchfield - Computer Sciences Corporation
Systems Sciences Division
4600 Powder Mill Road
Beltsville, MD 20705

B. Das - Computer Sciences Corporation
Systems Sciences Division
8728 Colesville Road
Silver Spring, MD 20910

T. Davis - NASA Kennedy Space Center
Cape Canaveral, FL 32899

L. Gilstrap - Computer Sciences Corporation
Systems Sciences Division
8728 Colesville Road
Silver Spring, MD 20910

R. Jackson - Space Telescope Science Institute
3700 San Martin Drive
Baltimore, MD 21218

C. Knoblock - Carnegie Mellon University
Department of Computer Science
5000 Forbes Ave
Pittsburgh, PA 15213

K. Lindenmayer - Space Telescope Science Institute
3700 San Martin Drive
Baltimore, MD 21218

E. Medeiros - Space Telescope Science Institute
3700 San Martin Drive
Baltimore, MD 21218

G. Miller	- Space Telescope Science Institute 3700 San Martin Drive Baltimore, MD 21218
K. Richardson	- Systems Autonomy Demonstration Project Office NASA/Ames Research Center Moffet Field, CA 94035
A. Weiss	- The MITRE Corporation 7525 Colshire Drive McLean, VA 22102
B. Woodard	- Computer Sciences Corporation Systems Division 3160 Fairview Park Drive Falls Church, VA 22042

In addition, the task members talked briefly with Harry Lum at NASA/Ames, R. Brown at Johnson Space Center, Nancy Martin of SoftPert Systems, and Ann Baker of CSC at Johnson Space Center.

APPENDIX D - CORE INTERVIEW QUESTIONS

The following represents the core set of questions used as a base for the interviews. Depending on the nature of the responses in a given interview, some questions will have been omitted and others delved into more deeply.

I. PERSONAL

Title

Company

Type of work company is generally involved in

Current responsibility (e.g., management/technical)

Experience in software development

Experience in expert system development

II. TECHNICAL

If interviewee is not currently working on an expert system project, direct questions to the project with which interviewee is most familiar.

A. PROJECT INFORMATION

Describe project in terms of

- Purpose
- Current status (e.g., what phase of development)
- Size (staff and system)
- Type (e.g., a research expert system or an operational expert system; deep knowledge as opposed to a quick problem solver)

B. METHODOLOGY

Does your company have an established methodology for developing software? Describe briefly (waterfall method? 2167?).

Did you follow this methodology on your expert system task? If not, what took its place?

What tools, if any, have you used?

Why did you choose it?
Did you like it? Was it satisfactory? How could it be improved?

How was (do you plan to) knowledge acquire(d)?

How long did it take?
Did you talk to an expert?
How often did the knowledge engineer/expert meet?
Was the expert part of the project team?

Did you use (rapid) prototyping?

Alone or in combination with some other approach?
When did you stop using it?

How did you evaluate (test) the expert system?

What QA/CM activities normally occur in software development development in your company's methodology?

Software development notebooks
Inspections
Checklists
Baselines
Configuration control
Status accounting

Were any of these applied to expert system development?

If so, which ones?
Did they differ from the regular QA/CM methods? If so, how?

How did you document the expert system, especially during prototyping?

C. SCHEDULE

How long has project been in existence?

How much more to go?

Who developed the schedule (management or technical or both)?

Did you use a scheduling tool? If so, which one?

Any problems meeting schedule, i.e., is schedule realistic?

D. MANAGEMENT APPROACH

What mechanisms were used to

- Perform the task planning
- Determine budget (staffing, not dollars)
- Monitor and control the task (e.g., reviews and demonstrations; milestones?)
- Report status
- Quantify work

E. PROBLEMS

What problems did you encounter?

- Performance?
- Communication (expert/knowledge engineer, task members, technical staff/management)?
- User satisfaction?
- Maintenance?
- Schedule/budget?
- Tools/development environment choice?
- Other?

If none, what did you do that worked so well?

GLOSSARY

AI	artificial intelligence
CSC	Computer Sciences Corporation
DEC	Digital Equipment Corporation
DSDM	Digital System Development Methodolgy
ESCIE	Expert System Controlled Iterative Enhancement
GSFC	Goddard Space Flight Center
IEEE	Institute of Electronic and Electrical Engineers
KBS	knowledge-based system
MO&DSD	Mission Operations and Data Systems Directorate
NASA	National Aeronautics and Space Administration
PMS	Platform Management System
PRESS	PMS Resource Envelope Scheduling System
SSD	System Sciences Division
ST ScI	Space Telescope Science Institute

REFERENCES

1. D. A. Waterman, A Guide to Expert Systems, Reading, Mass.: Addison-Wesley, 1986

2. L. G. Hull, "Managing an Expert System Project" (paper presented at the Goddard Space Flight Center 1986)

3. B. Buchanan, D. Barstow, R. Bechtel, J. Bennett, W. Coancey, C. Kulikowski, T. Mitchell, and D. A. Waterman, "Constructing an Expert System", Building Expert Systems, F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, eds. Reading, Mass.: Addison-Wesley, pp. 127-167, 1983

4. B. Hancock, "Expert Systems," DEC Professional, vol. 6, no. 5, pp. 40-48, 1987

5. R. Keller, Expert System Technology: Development and Application, Englewood Cliffs, N.J.: Yourdon Press, 1987

6. B. K. Das and R. A. Berg, "Impact Assessment: Perspective of an Operator Assistant for Scheduling NASA's Tracking and Data Relay Satellite System" (poster session at the Third IEEE Conference on Artificial Intelligence Applications, Orlando, FL, February 1987), pp. 23-27

7. B. K. Das and R. A. Berg, "Expert System: A Network Operator Assistant (NOA)" (CSC presentation 1987)

8. A. T. Goldberg, "Knowledge-based Programming: A Survey of Program Design and Construction Techniques," IEEE Transactions on Software Engineering, vol. SE-12, no. 7, pp. 752-768, 1986

9. K. Lindenmayer, S. Vick, and D. Rosenthal, "Maintaining an Expert System for the Hubble Space Telescope Ground Support," Proceedings of the Goddard Conference on Space Applications of AI and Robotics, 1987

10. P. Friedland, "Architectures for Knowledge-Based Systems," Communications of the ACM, vol. 28, no. 9, p. 903, 1985

11. P. J. Denning, "Towards a Science of Expert Systems," IEEE Expert, vol. 1, no. 2, pp. 80-83, 1986

12. A. Kraft, "XCON: An Expert Configuration System at Digital Equipment Corporation," The AI Business, P.H. Winston and K. A. Prendergast, eds. London, England: MIT Press, pp. 41-49, 1984

13. J. S. Brown, "The Low Road, the Middle Road, and the High Road," The AI Business, P.H. Winston and K. A. Prendergast, eds. London, England: MIT Press, pp. 81-90, 1984

14. J. Gaschnig, P. Klahr, H. Pople, E. Shortliffe, and A. Terry, "Evaluation of Expert Systems: Issues and Case Studies," Building Expert Systems, F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, eds. Reading, Mass.: Addison-Wesley, pp. 241-280, 1983

15. H. A. Simon, "Whether Software Engineering Needs to Be Artificially Intelligent," IEEE Transactions on Software Engineering, vol SE-12, no. 7, pp. 726-732, 1986

16. "Interview: Peter Hart Talks about Expert Systems," IEEE Expert, vol. 1, no. 1, pp. 96-99, 1986

17. F. Hayes-Roth, "The Knowledge-Based Expert System: A Tutorial," Computer, vol. 17, no. 9, pp. 11-28, 1984

18. W. Myers, "Introduction to Expert Systems," IEEE Expert, vol. 1, no. 2, pp. 100-109, 1986

19. J. H. Boose, "Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain," Second Conference on Artificial Intelligence Applications, Miami Beach, December 11-13, 1984, pp. 461-466

20. N. F. Schneidewind, "The State of Software Maintenance," IEEE Transactions on Software Engineering, vol. SE-13, no. 3, pp. 303-310, 1987

21. K. Narayanaswamy and W. Scacchi, "Maintaining Configurations of Evolving Software Systems," IEEE Transactions on Software Engineering, vol. SE-13, no. 3, pp. 324-334, 1987

22. R. Balzer, "A 15 Year Perspective on Automatic Programming," IEEE Transactions on Software Engineering, vol. SE-11, no. 11, pp. 1257-1268, 1985

23. K. Swigger, H. Burns, H. Loveland, T. Jackson, "An Intelligent Tutoring System for Interpreting Ground Tracks," Proceedings of the Sixth National Conference on AI, Seattle, WA, July 13-17, 1987, vol. 1, pp. 72-76

24. B. Woolf and P. Cunningham, "Building A Community for Intelligent Tutoring Systems ," Proceedings of the Sixth National Conference on AI, Seattle, WA, July 13-17, 1987, vol. 1, pp. 82-87

25. V. E. Kelly and U. Nonnenmann, "Inferring Formal Software Specifications From Episodic Descriptions," Proceedings of the Sixth National Conference on AI, Seattle, WA, July 13-17, 1987, vol. 1, pp. 127-132

26. C. Hayes, "Using Goal Interactions to Guide Planning", Proceedings of the Sixth National Conference on AI, Seattle, WA, July 13-17, 1987, vol. 1, pp. 224-228

27. A. L. Lansky and D. S. Fogelsong, "Localized Representation and Planning Methods for Parallel Domains,"Proceedings of the Sixth National Conference on AI, Seattle, WA, July, 13-17, 1987, vol. 1, pp. 240-245

28. R. Kass and T. Finin, "Rules for the Implicit Acquisition of Knowledge about the User," Proceedings of the Sixth National Conference on AI, Seattle, WA, July 13-17, 1987, vol. 1, pp. 295-300

29. T. C. Walker and R. K. Miller, Expert Systems 1986, Madison, GA: SEAI Technical Publications, 1986

30. A. Ram, "AQUA: Asking Questions and Understanding Answers," Proceedings of the Sixth National Conference on AI, Seattle, WA, July 13-17, 1987, vol. 1, pp. 312-316

31. P. A. Subrahmanyam, "The 'Software Engineering' of Expert Systems: Is Prolog Appropriate?," IEEE Transactions on Software Engineering, vol. SE-11, no. 11, pp. 1391-1400, 1985

32. D. G. Bobrow, "If Prolog Is The Answer, What is the Question? or What It Takes to Support the AI Programming Paradigms," IEEE Transactions on Software Engineering, vol. SE-11, no. 11, p. 1401, 1985

33. R. Fikes and T. Kehler, "The Role of Frame-Based Representation in Reasoning," Communications of the ACM, vol. 28, no. 9, pp. 904-920, 1985

34. M. R. Genesereth and M. L. Ginsberg, "Logic Programming," Communications of the ACM, vol. 28, no. 9, pp. 933-941, 1985

35. J. F. Brule, "Expert Systems: Applications," Artificial Intelligence: Theory, Logic and Application, Blue Ridge Summit, PA: Tab Books, Inc., 1986

36. D. C. Bochsler and M. A. Goodwin, "A Software Engineering Approach to Expert System Design and Verification," Conference on Artificial Intelligence for Space Applications, Huntsville, AL, November 13-14, 1986, p. 47

37. E. Medeiros, Steps to a New Methodology (unpublished)

38. An untitled presentation by personnel from Inference

39. B. K. Das and R. A. Berg, "Impact Assessment: A Case Study for Scheduling NASA's TDRSS Resources," Proceedings of Third Annual Expert Systems in Government Conference, Washington, DC, October 19-23, 1987, pp. 254-259

40. D. G. Bobrow, S. Mittal, M. J. Stefik, "Expert Systems: Perils and Promise," Communications of the ACM, vol. 29, no. 9, pp. 880-894, 1986

41. E. Soloway, J. Bachant, and K. Jensen, "Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY Large Rule-Base," Proceedings of the Sixth National Conference on AI, Seattle, WA, July 13-17, 1987, vol. 1, pp. 824-829

42. F. Hayes-Roth, "Rule-Based Systems," Communications of the ACM, vol. 28, no. 9, pp. 921-932, 1985

43. G. Ringland, "Software Engineering in a Development Group," Software-Practice and Experience, vol. 14, no. 6, pp. 533-559, 1984

44. R. J. K. Jacob and J. N. Froscher, Developing a Software Engineering Methodology for Knowledge-Based Systems, Naval Research Laboratory Report 9019, 1986

45. Mission Operations and Data Systems Directorate Systems Management Policy, NASA, Goddard Space Flight Center, MDOD-8YMP/0485, 1986

46. Mission Operations and Data Systems Directorate Software Development Policy, NASA, Goddard Space Flight Center, MODSD-8YDP/0186, 1986

47. S. Steppel, T. L. Clark, P. C. Belford, W. D. Bumgarner, A. M. Federoff, N. S. Frankle, J.A. Torreele, and J. A. Zaccaro, Digital System Development Methodology (DSDM), Computer Sciences Corporation, 1985

48. W. Teitelman, "A Tour Through Cedar," IEEE Transactions on Software Engineering, vol. SE-11, no. 3, pp. 285-302, 1985

49. I. Zualkernan, W. T. Tsai, and D. Volovik, "Expert Systems and Software Engineering: Ready for Marriage?," IEEE Expert, vol. 1, no. 4, pp. 24-31, 1986

50. M. D. Lubars and M. T. Harandi, "Intelligent Support for Software Specification and Design," IEEE Expert, vol. 1, no. 4, pp. 33-41, 1986

51. R. P. Frail and R. S. Freedman, "OPGEN Revisited: Some Methodological Observations on the Delivery of Expert Systems," Expert Systems in Government Symposium, McLean, VA, October 22-24, 1986

52. R. Jackson, "Expert Systems Built by the 'Expert': An Evaluation of OPS5," Proceedings of the Goddard Conference on Space Applications of AI and Robotics, 1987

53. D. Partridge, "Engineering Artificial Intelligence Software," Artificial Intelligence Review, vol. 1, pp. 27-41, 1986

54. J. M. Fox, "AI's Contribution to Software Development or Expert Systems Are Symptoms," Expert Systems in Government Symposium, McLean, VA, October 22-24, 1986

55. W. C. Dias, J. A. Henricks, and J. C. Wong, "PLAN-IT: Scheduling Assistant for Solar System Exploration," Proceedings of the Goddard Conference on Space Applications of AI and Robotics, 1987

56. K. M. Lines-Browning and J. L. Stone, "An Expert System That Performs A Satellite Stationkeeping Maneuver," Proceedings of the Goddard Conference on Space Applications of AI and Robotics, 1987

57. T. Bollinger, E. Lightner, J. Laverty, and E. Ambrose, "The Load Shedding Advisor: An Example of a Crisis-Response Expert System," Proceedings of the Goddard Conference on Space Applications of AI and Robotics, 1987

58. R. Dutilly, "Automation of Spacecraft Control Centers," Proceedings of the Goddard Conference on Space Applications of AI and Robotics, 1987

59. J. P. Jenkins and W. R. Nelson, "Expert Systems and Accident Management," Expert Systems in Government Symposium, McLean, VA, October 22-24, 1986, pp. 88-95

60. D. Brauer, P. Roach, M. Frank, and R. Knackstedt, "Ada and Knowledge Base Systems: A Prototype Combining the Best of Both Worlds," Expert Systems in Government Symposium, McLean, VA, October 22-24, 1986, pp.198-202

61. F. W. Rook and J. B. Odubiyi, "An Expert System for Satellite Orbit Control (ESSOC)," Expert Systems in Government Symposium, McLean, VA, October 22-24, 1986, pp. 221-233

62. D. W. Rolston, "An Expert System for Reducing Software Maintenance Costs," Expert Systems in Government Symposium, McLean, Va, October 22-24, 1986, pp. 396-406

63. J. A. Robinson and A. A. Satterlee, "An In-Service Expert System for Configuring Thrusters on Orbiting Spacecraft," Proceedings of the Western Conference on Expert Systems, Anaheim, CA, June 2-4, 1987, pp. 112-117

64. R. M. O'Bannon, "An Intelligent Aid to Assist Knowledge Engineers with Interviewing Experts," Proceedings of the Western Conference on Expert Systems, Anaheim, CA, June 2-4, 1987, pp. 31-36

65. C. M. Kitto and J. H. Boose, "Choosing Knowledge Acquisition Strategies for Application Tasks," Proceedings of the Western Conference on Expert Systems, Anaheim, CA, June 2-4, 1987, pp. 96-103

66. C. J. R. Green and M. M. Keyes, "Verification and Validation of Expert Systems," Proceedings of the Western Conference on Expert Systems, Anaheim, CA, June 2-4, 1987, pp. 38-43

67. M. Ebrahimi, "A Structured Approach to Expert System Design," Proceedings of the Western Conference on Expert Systems, Anaheim, CA, June 2-4, 1987, pp. 18-24

68. K. Lantz, "The Prototyping Methodology: Designing Right the First Time," Computerworld, pp. 69-72, April 7, 1986

69. G. R. Gladden, "Stop the Life-Cycle, I Want to Get Off," ACM SIGSOFT Software Engineering Notes, vol. 7, no. 2, pp. 35-39, 1982

70. R. Balzer, T. E. Cheatham, and C. Green, "Software Technology in the 1990's: Using a New Paradigm," IEEE Computer, vol. 16, no. 11, pp. 39-45, 1983

71. T. Gilb, "Evolutionary Delivery versus the "Waterfall Model"," ACM SIGSOFT Software Engineering Notes, vol. 10, no. 3, pp. 49-61, 1985

72. D. Leinweber, "Expert Systems in Space," IEEE Expert, vol.2, no. 1, pp. 26-36, 1987

73. A. van de Brug, J. Bachant, and J. McDermott, "The Taming of R1," IEEE Expert, vol. 1, no. 3, pp. 33-39, 1986

74. R. Kowalski, "AI and Software Engineering," Datamation, vol. 30, no. 18, pp. 92-102, 1984

75. C. Ramamoorthy, A. Prakash, W. Tsai, and Y. Usuda, "Software Engineering: Problems and Perspectives," Computer, vol. 17, no. 10, pp. 191-209, 1984

76. P. A. Bailey, and B. B. Doehr, "Knowledge Acquisition and Rapid Prototyping of an Expert System: Dealing with 'Real World' Problems," Conference on Artificial Intelligence for Space Applications, Huntsville, AL, November 13-14, 1986

77. W. W. Agresti, "The Conventional Software Life-cycle Model: Its Evolution and Assumptions," New Paradigm for Software Development, IEEE Computer Society Press, Washington, D. C., 1986

78. "Defense Science Board Final Report on the Software Task Force," 1987

79. Defense System Software Development, DOD-STD-2167, Department of Defense, 1985

80. J. Connell, Software Rapid Prototyping seminar, August 27-28, 1987, Washington, DC

81. T. DeMarco, Structured Analysis and System Specification, Yourdon, Inc., New York 1979

82. P. J. Kline and S. B. Dolins, "Choosing Architectures for Expert Systems," US Department of Commerce, NTIS, Springfield, VA, 1985

83. B. W. Boehm, Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliff, NJ, 1981

84. B. A. Zack, "Building Operational Expert Systems," Tutorial presented at Third Annual Expert Systems in Government Conference, Washington, DC, October, 1987

85. N. Martin, "The Management of Expert System Development," Tutorial presented at Third Annual Expert Systems in Government Conference, Washington, DC, October, 1987

86. F. P. Brooks Jr., "No Silver Bullets: Essence and Accidents of Software Engineering," Computer, vol. 20, no. 4, pp. 10-19, 1987

87. T.C. Hartrum and G. B. Lamont, "Development of a Comprehensive Software Engineering Environment," presented at Space Operations and Robotics '87 Workshop, Johnson Space Center, Houston, TX, August, 1987

88. B. P. Allen and P. L. Holtzman, "Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project," presented at the Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX, August, 1987

89. W. B. Wingert, "Verifying Shuttle Onboard Software Using Expert Systems," presented at the Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX, August, 1987

90. R. A. Stachowitz, C. L. Chang, T. S. Stock, and J. B. Combs, "Building Validation Tools for Knowledge-Based Systems," presented at the Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX, August, 1987

91. M. A. Goodwin and C. C. Robertson, "Expert System Verification Concerns in an Operations Environment," presented at the Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX, August, 1987

92. B. Kozick Jr. and W. Reynolds, "MPAD MCC Workstation System: ADDAM and EEVE," presented at the Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX, August, 1987

93. C. Culbert, G. Riley, and R. T. Savely, "Approaches to the Verification of Rule-Based Expert Systems," presented at the Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX, August, 1987

94. G. Castore, "A Formal Approach to Validation and Verification of Knowledge-Based Control Systems," presented at the Space Operations Automation and Robotics (SOAR) '87 Workshop, Johnson Space Center, Houston, TX, August, 1987

95. C. Culbert, G. Riley, and R. T. Savely, "An Expert System Development Methodology which Supports Verification and Validation," Submitted to 4th IEEE Conference on Artificial Intelligence, 1987

96. R. L. Citrenbaum and J. R. Geissman, "A Practical Cost-Conscious Expert System Development Methodology," presented at AI-86: Artificial Intelligence and Advance Computer Technology Conference, Long Beach, CA, April 29 - May 1, 1986

97. R. A. Stachowitz and J. B. Combs, "Validation of Expert Systems," Proceedings of the Hawaii International Conference on Systems Sciences, Kona, Hawaii, 1987

98. R. M. O'Keefe, O. Balci, and E. P. Smith, "Validating Expert System Performance," IEEE Expert, vol. 2, no. 4, pp. 81-90, 1987

99. M. Patrick, "Surprise Control," Computerworld, vol. 21, pp. 93-98, November 16, 1987

100. P. J. Kline and S. B. Dolins, "Problem Features That Influence the Design of Expert Systems," AAAI 86: Fifth National Conference on Artificial Intelligence, pp. 956-962, 1986

101 D. S. Prerau, "Knowledge Acquisition in the Development of a Large Expert System," AI Magazine, vol. 8, no. 2, pp. 43-51, 1987

102. T. A. Nguyen, W. A. Perkins, T. J. Laffey, and D. Pecora, "Knowledge Base Verification," AI Magazine, vol. 8, no. 2, pp. 69-75, 1987

103. K. Richardson and C. Wong, "Knowledge Based System Verification and Validation as Related to Automation of Space Station Subsystems: Rationale for a Knowledge Based System Lifecycle," presented at Third Conference on Artificial Intelligence for Space Applications, Huntsville, AL, November 2-3, 1987

104. Proceedings of the Third Conference on Artificial Intelligence for Space Applications, NASA Conference Publication 2492, Huntsville, AL, November 2-3, 1987